



8-28-00

A

Customer No. 20350
TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, 8th Floor
San Francisco, California 94111-3834
(415) 576-0200

08/24/00

Attorney Docket No. 17887-003500US

"Express Mail" Label No. EL623997210US

Date of Deposit: August 24, 2000



ASSISTANT COMMISSIONER FOR PATENTS
BOX PATENT APPLICATION
Washington, D.C. 20231

Sir:
Transmitted herewith for filing under 37 CFR 1.53(b) is the

[X] patent application of

Inventor(s)/Applicant Identifier: Udi Manber et al.

For: SYSTEMS AND METHODS FOR IDENTIFYING AND EXTRACTING DATA FROM HTML PAGES

Enclosed are:

- [X] 11 page(s) of specification.
[X] 4 page(s) of claims.
[X] 1 page of Abstract.
[X] 2 sheet(s) of [] formal [X] informal drawing(s).
[X] APPENDIX A (83 pages).
[X] Declaration (unsigned).

(Col. 1) (Col. 2)

FOR:	NO. FILED	NO. EXTRA
BASIC FEE		
TOTAL CLAIMS	25 - 20	= *5
INDEP. CLAIMS	4 - 3	= *1
[] MULTIPLE DEPENDENT CLAIM PRESENTED		

* If the difference in Col. 1 is less than 0, enter "0" in Col. 2.

SMALL ENTITY

RATE	FEE
x \$9.00 =	\$345.00
x \$39.00 =	
+ \$130.00 =	
TOTAL	

OTHER THAN SMALL ENTITY

RATE	FEE
x \$18.00 =	\$690.00
x \$78.00 =	\$78.00
+ \$260.00 =	
TOTAL	\$858.00

Please charge Deposit Account No. 20-1430 as follows:

- [X] Filing fee \$ \$858.00
[X] Any additional fees associated with this paper or during the pendency of this application.
[] The issue fee set in 37 CFR 1.18 at or before mailing of the Notice of Allowance, pursuant to 37 CFR 1.311(b)

- [] A check for \$ is enclosed.
2 extra copies of this sheet are enclosed.

Respectfully submitted,
TOWNSEND and TOWNSEND and CREW LLP

Gerald T. Gray
Gerald T. Gray
Reg No.: 41,797
Attorneys for Applicant

Telephone: (415) 576-0200
Facsimile: (415) 576-0300

PATENT APPLICATION

**SYSTEMS AND METHODS FOR IDENTIFYING AND
EXTRACTING DATA FROM HTML PAGES**

Inventors: Udi Manber, residing at
883 Robb Road
Palo Alto, CA 94306,
a citizen of The United States of America

Qi Lu, residing at
20847 Russell Lane
Saratoga, CA 95070
a citizen of The United States of America

Assignee: YAHOO, INC.
3420 Central Expressway
Santa Clara, CA 95051

Entity: Large

SYSTEMS AND METHODS FOR IDENTIFYING AND EXTRACTING DATA FROM HTML PAGES

5

COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

The present invention relates generally to analyzing and extracting information from web pages, and more particularly to automatically identifying and extracting desired information in web pages.

The World Wide Web (WWW) is now the premier outlet to publish information of all types and forms. Documents published on the web, commonly called web pages, are published using a language called HTML (or Hyper Text Markup Language), which sets standards for the formatting of documents. These standards make it possible for people to read and understand documents no matter which program they use for that purpose. For the most part, documents are designed and written to be read by real persons. But there is a growing need to have automatic programs extract certain parts of documents with minimal human intervention. For example, suppose that a document D contains information about product P. D may contain a picture of P, its description, its price, its availability and several characteristics of P. A different document D', published by a different company about the same product P, may have similar parts, but they will most likely be arranged and formatted in a completely different way. People reading D and D' can easily parse the information and understand its different pieces, but it is difficult for a computer program to do so without knowing in advance which pieces are included and how they are arranged. The same company that published the web page for

product P may also publish pages on numerous other products. These pages may be similarly formatted, but since they describe different products they contain entirely different information.

- 5 As an example, a typical HTML document includes formatting commands or tags, and content which can be text, images, programs, and so on. HTML tags are enclosed in brackets <>. For example, the text

“Product P available in California ON SALE for \$19.99”

can be formatted as:

```

10      <table>
      <tr>
      Product P
      <img src=p.gif>
      <i>available</i>
      in California
15      <font color=red>
      ON SALE
      </font>
      for $19.99
      </tr>
20      </table>
  
```

This HTML code puts the line as a row in a table, adds an image, italicizes “available”, and highlights “ON SALE” in red. A typical commerce page may have hundreds of formatting tags.

- 25 A different product Q may appear as:

“Product Q in Oregon and Washington for \$15.99”

and be formatted as:

```

      <table>
      <tr>
30      Product Q
      <img src=q.gif>
      <i>available</i>
      in Oregon and Washington for $15.99
  
```

</tr>

</table>

5 If one is interested in extracting only the price of the product, a typical rule-based extraction mechanism, using the first document for product P, may infer that the price appears after the ON SALE text, or after the red formatted text. However, this same extraction mechanism, when analyzing the second document for product Q, will miss the price of product Q, because neither the ON SALE text nor the red formatting is present. In general, the page may be much more complex and variable.

10 Accordingly, it is desirable to provide methods and systems for analyzing the structure of web pages and for automatically extracting pertinent information from the web pages.

15

SUMMARY OF THE INVENTION

The present invention provides systems and methods for analyzing web pages formatted using HTML or other markup language to automatically identify and extract desired information. In one embodiment, aspects of the invention are embodied in a computer algorithm that identifies and extracts different pieces of information from different web pages automatically after minimal manual setup. The algorithm automatically analyzes pages with different content if they have the same, or similar, formats. The algorithm is robust, in the sense that it operates successfully and correctly in the presence of small changes to the formatting of documents. The algorithm is fast and efficient and performs the extraction process quickly in real-time. Many database and data mining applications require structured data -- they have to know the meanings of numbers and text, and not just their values, so they can infer relationships among them. Using the techniques of the present invention, it becomes possible to build databases from unstructured web information. The algorithm can be implemented in an agent that captures information about products, and compares prices or other characteristics. The algorithm can also be used to populate structured databases that, given the different pieces of information, can analyze products and their characteristics. Additionally, the algorithm

can be used for data mining applications, e.g., looking for patterns useful for marketing analyses, for testing and quality assurance (QA) purposes, or other uses.

- According to an aspect of the invention, a method is provided for identifying and extracting content from HTML formatted web pages. The method typically comprises the steps of selecting a model page, wherein the model page includes a plurality of HTML tags, identifying an area of interest in the model page, and parsing the model page to determine a first string of symbols associated with the plurality of HTML tags, wherein the first area of interest is identified by a first portion of the first string of symbols. The method also typically includes the steps of retrieving a second web page, parsing the second web page to determine a second string of symbols associated with the HTML tags of the second web page, comparing the first and second strings to determine whether the second string includes a second portion similar to the first portion of the first string, wherein the second portion corresponds to a second area of interest in the second page, and thereafter extracting the second area of interest from the second page. In preferred aspects the steps of selecting the model page and identifying a first area of interest are performed manually, and the remaining steps are performed automatically.

- According to another aspect of the present invention, a computer readable medium is provided containing instructions for controlling a computer system to automatically identify and extract desired content from a retrieved HTML formatted web page. The medium includes instructions to control the computer system to automatically parse the HTML code of a manually selected model web page to determine a first string of symbols associated with a first plurality of HTML tags. The medium also typically includes instructions to control the computer system to automatically retrieve a second web page, parse the HTML code of the second web page to determine a second string of symbols associated with HTML tags of the second page, compare the first and second strings to determine whether the second page includes a second plurality of HTML tags substantially matching the first plurality of HTML tags, and extract a portion of the second page corresponding to the second plurality of HTML tags.

- According to yet another aspect of the present invention, a computer system is provided for identifying and extracting content from HTML formatted web pages. The system typically comprises a means for retrieving web pages including HTML tags, wherein a model web page is retrieved, and a means for manually identifying

a first area of interest in the model page, wherein the first area of interest corresponds to a first plurality of HTML tags. The system also typically comprises a processor including a means for parsing a page, wherein the parsing means parses the model page to determine a first string of symbols associated with the first plurality of HTML tags, and wherein the parsing means thereafter parses an automatically retrieved second web page to determine a second string of symbols associated with the HTML tags of the second web page. The processor also typically includes a means for comparing the first and second strings to determine whether the second string includes a second portion similar to the first portion of the first string, wherein the second portion corresponds to a second area of interest in the second page, and a means for extracting the second area of interest from the second page.

According to a further aspect of the invention, a computer implemented method of identifying and extracting content from web pages formatted using a markup language is provided. The method typically includes the steps of selecting a model page, wherein the model page includes a plurality of tokens, identifying a first area of interest in the model page, and parsing the model page to determine a first string of symbols associated with the plurality of tokens, wherein the first area of interest is identified by a first portion of the first string of symbols. The method also typically includes the steps of retrieving a second web page, parsing the second web page to determine a second string of symbols associated with the tokens of the second web page, comparing the first and second strings to determine whether the second string includes a second portion similar to the first portion of the first string, wherein the second portion corresponds to a second area of interest in the second page, and thereafter extracting the second area of interest from the second page. The present invention is applicable to any markup language, including any instance of SGML, such as XML, WML, HTML, DHTML and HDML.

Reference to the remaining portions of the specification, including the drawings and claims, will realize other features and advantages of the present invention. Further features and advantages of the present invention, as well as the structure and operation of various embodiments of the present invention, are described in detail below with respect to the accompanying drawings. In the drawings, like reference numbers indicate identical or functionally similar elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a general overview of an information retrieval and communication system according to an embodiment of the present invention; and

Figure 2 is a flow chart showing the process of identifying and extracting information from web pages according to an embodiment of the present invention.

DESCRIPTION OF THE SPECIFIC EMBODIMENTS

Figure 1 illustrates a general overview of an information retrieval and communication network 10 including a client device 20 according to an embodiment of the present invention. In computer network 10, client device 20 is coupled through the Internet 40, or other communication network, to servers 50₁ to 50_N. Client device 20 is also interconnected to server 30 either directly, over any LAN or WAN connection, or over the Internet 40. As will be described herein, client device 20 is configured according to the present invention to access and retrieve web pages from any of servers 50₁ to 50_N, identify and extract desired information therefrom, and provide the information to server 30 to populate database 35. Although as described herein, access and processing of web pages is performed using client device 20, it will be understood that server 30 can also be configured to access and process web pages according to the present invention described herein.

Several elements in the system shown in Figure 1 are conventional, well-known elements that need not be explained in detail here. For example, client device 20 (and server 30) could be a desktop personal computer, workstation, laptop, PDA, cell phone, or any WAP-enabled device or any other computing device capable of interfacing directly or indirectly to the Internet. Client device 20 typically runs a browsing program, such as Microsoft's Internet Explorer, Netscape Navigator or the like, allowing a user of client 20 to access and browse pages available to it from servers 50₁ to 50_N over Internet 40. Client 20 (and server 30) also typically includes one or more user interface devices 22, such as a keyboard, a mouse, touchscreen, pen or the like, for interacting with a graphical user interface (GUI) provided by the browser in conjunction with pages and forms retrieved from servers 50₁ to 50_N or other servers. The present invention is suitable for use with the Internet, which refers to a specific global Internetwork of networks. However, it should be understood that other networks can be used instead of the Internet,

such as an intranet, an extranet, a virtual private network (VPN), a non-TCP/IP based network, or the like.

According to one embodiment, client device 20 or server 30, and all of its components are operator configurable using an application including computer code run using a central processing unit such as an Intel Pentium processor or the like. Computer code for operating and configuring client device 20 or server 30 as described herein is preferably stored on a hard disk, but the entire program code, or portions thereof, may also be stored in any other volatile or non-volatile memory medium or device as is well known, such as a ROM or RAM, or provided on any media capable of storing program code, such as a compact disk medium, DVD, a floppy disk, or the like. Additionally, the entire program code, or portions thereof, may be downloaded from a software source to client device 20 or server 30 over the Internet as is well known, or transmitted over any other conventional network connection as is well known, e.g., extranet, VPN, LAN, etc., using any communication medium and protocol as are well known. Appendix A includes an example of code for implementing the techniques of the present invention. It will also be appreciated that computer code for implementing the present invention can be implemented in JavaScript, or any scripting language such as VBScript, that can be executed on a client device or server system. Although it is understood that server 30, or any other server, can be configured using the code as above, the following will discuss the present invention implemented in the context of client device 20.

In general, a user is able to access and query servers 50_1 to 50_N and other servers through client device 20 to view and download content such as news stories, advertising content, search query results including links to various websites and so on. Such content can also include other media objects such as video and audio clips, URL links, graphic and text objects such as icons and hyperlinks, and the like. As described herein, the techniques of the present invention are particularly useful for identifying and extracting information related to products from remote vendor servers. Such information can be used, for example, to populate database 35 with comparative information for access by subscribers or the general public, e.g., over the Internet. For example, the extracted information can be used to populate database 35 with comparative pricing information for a particular product or service or related products or services. One example of such an accessible server/database for which the invention is useful is the Yahoo! Shopping website located at <http://shopping.yahoo.com>. It will of course be

apparent that the present invention is useful for identifying and extracting any desired information in web pages retrieved from any website for use in any data mining application or other application.

Figure 2 is a flow chart showing the process of identifying and extracting information from web pages according to an embodiment of the present invention. In the following description, it is assumed that the web pages are formatted using HTML, although the present invention is equally applicable to processing web pages formatted using any markup language including any instance of the Standard Generalized Markup Language (SGML), such as XML, WML, HDML (for hand-held devices), DHTML and others.

According to one embodiment, at step 100 an operator using client device 20 (or server 30) first selects a target page that is deemed a model page for a particular product type, company format, or any other type of document. For example, the operator accesses a particular product page for product P from one of servers 50₁ to 50_N, which corresponds to a particular remote vendor's website. At step 110, the HTML code for the selected page is parsed to determine a model pattern for the page. In one embodiment, a model pattern based on the selected page is built by first dividing the web page into HTML tokens. In general, HTML tokens include tag elements and text elements. In one embodiment, the text is preferably initially ignored, and the tags that are primarily used for formatting purposes, e.g., <form>, , rather than being a major part of the design of pages, are also preferably ignored (which tokens to ignore is an option set by the operator in one embodiment). The remainder is typically a sequence of start tags and closing tags. Each of the tokens is preferably translated into a unique number, represented for illustration purposes as a character. For example, the format for product P above can be represented as a sequence TRGIF/F/R/T, where T represents <table>, /T represents </table> (end table), R represents <tr>, and so on. In this representation scheme, the HTML code for product Q would be represented as a sequence TRGI/R/T.

In general HTML includes the name of the tag, e.g., <table>, and also several possible attributes, e.g., . In preferred aspects, the present invention is configured to parse the HTML document using the tags and the attributes as will be discussed below in more detail.

At step 120, the operator identifies an area of interest in the selected page. In one embodiment, a graphical user interface (GUI) is provided to the operator as part of

a manual selection and extraction tool. The operator is able to select or highlight portions of the page that are of interest, e.g., the price of product P and/or the red-formatted ON SALE text. The operator preferably selects portions of the page (e.g., portions of the displayed web page or portions of the corresponding HTML code) using interface device

22, such as a mouse or keyboard or other manual pointing and selecting device. The operator can select to ignore parts of the pattern sequence. For example, the beginning of the document may include text not directly related to the desired information (e.g., ads). The operator can choose to ignore this information, for example, by not selecting this portion with the selection tool. That which is selected (i.e., not ignored) is stored as the model pattern sequence at step 130. It will be apparent that steps 110 and 120 are interchangeable in that the operator can first select the desired area in the selected page and thereafter the application will parse the HTML code corresponding to the selected area of the page. At step 130, the model pattern is stored in a memory for comparison with patterns representing portions of other pages.

At step 140, another page (e.g., related document) is retrieved from the same vendor site or from a different site. Client 20 is preferably configured to automatically retrieve a subsequent page for analysis from a website using the site's URL. For example, client 20 may be configured to retrieve a subsequent page from the same site from which the target page was retrieved, or from a list of one or more universal

20 resource locators (URLs), either randomly or in a specific order. This subsequent page is then parsed to produce a pattern sequence for comparison with the stored pattern of the target page to identify related information. In particular, at step 150, the HTML code of this subsequent document is parsed as in step 110. At step 160, the stored model pattern sequence is compared against the pattern sequence obtained from the subsequent

25 document to identify matching or similar pattern segment(s). The two pattern sequences may not match exactly. According to one embodiment, an approximate string matching technique is used to solve this problem. Because the HTML code from both pages have been translated into sequences of characters (e.g., numbers) in steps 110 and 150, it is possible to employ an approximate pattern matching technique to match with high

30 confidence the parts of the patterns that correspond to the same type of information. Because HTML tokens can include attributes, it may not be sufficient to simply treat tokens as a character in the analysis. For example, the tags <table width = 50> and <table width = 100> represent two entirely different tables and should not be treated as the same.

Therefore, according to one embodiment of the present invention, the approximate string matching algorithm is extended to include comparisons of the attribute values. In the above example, comparisons of the attribute values for the table width tags are compared to determine whether a match exists. An example of an approximate string matching

5 algorithm for use with the present invention is included in Appendix A. At step 170, the results of the comparison are used to extract the desired information from the subsequent page to be stored (e.g., in database 35) and/or displayed. Any number of subsequent pages may be retrieved and analyzed with respect to the stored pattern of the target page by repeating steps 140 to 170.

10 According to another embodiment of the present invention, web pages are analyzed in a streaming fashion. There is no need to wait until the whole page is fetched for analysis. Whenever a part of the page is received it is analyzed immediately; that part of the page is parsed and compared with the stored pattern in real-time. If a matching pattern is found, the rest of the page can be discarded. Because the delays associated with

15 retrieving pages are usually more time consuming than the delays associated with executing the application program, a streaming approach speeds the process considerably.

Some web pages may have several alternative formats which are quite different and cannot be inferred from one to another. For example, results of a search for a particular author may return a list of authors matching the name, or a list of books by

20 the uniquely named author. According to one embodiment of the present invention, several patterns are analyzed in parallel. In this embodiment, all the model patterns that correspond to formats associated with a single site are compared, approximately, to the incoming page from that site. The first model pattern that leads to a good match is used, and the rest of the patterns may be ignored from then on.

25 Often a single page contains information about several products. A pattern used in this case can include several sub-patterns, each with a different use. One sub-pattern may identify the beginning of the list of items. Another may identify a pattern to ignore in the middle of the page. Yet another may identify a repeated item, for example, a list of books from the same authors, each with a description and a price. Accordingly it

30 is understood that the general approximate matching techniques as described herein enable the matching of different patterns for different purposes, all within the same framework.

To add to the reliability of the result, standard regular expression matching techniques are also applied to the extracted information in one embodiment. For example, when looking for a price, the result should contain a dollar sign (or other currency signs) and should be of a form that corresponds to a price. Such techniques are preferably used to help resolve cases where more than one piece matches the desired extracted information.

While the invention has been described by way of example and in terms of the specific embodiments, it is to be understood that the invention is not limited to the disclosed embodiments. To the contrary, it is intended to cover various modifications and similar arrangements as would be apparent to those skilled in the art. Therefore, the scope of the appended claims should be accorded the broadest interpretation so as to encompass all such modifications and similar arrangements.

WHAT IS CLAIMED IS:

1 1. A computer implemented method of identifying and extracting
 2 content from HTML formatted web pages, comprising the steps of:
 3 selecting a model page, wherein the model page includes a plurality of
 4 HTML tags;
 5 identifying a first area of interest in the model page;
 6 parsing the model page to determine a first string of symbols associated
 7 with the plurality of HTML tags, wherein the first area of interest is identified by a first
 8 portion of the first string of symbols;
 9 retrieving a second web page;
 10 parsing the second web page to determine a second string of symbols
 11 associated with the HTML tags of the second web page; and
 12 comparing the first and second strings to determine whether the second
 13 string includes a second portion similar to the first portion of the first string, wherein the
 14 second portion corresponds to a second area of interest in the second page.

1 2. The method of claim 1, wherein the step of comparing includes
 2 applying an approximate pattern matching algorithm to the first and second strings.

1 3. The method of claim 1, further comprising the step of storing the
 2 first and second areas of interest in a database.

1 4. The method of claim 1, further comprising the step of extracting
 2 the second area of interest from the second page.

1 5. The method of claim 4, further comprising the step of applying a
 2 regular expression matching algorithm to the extracted second area of interest.

1 6. The method of claim 1, wherein the first and second areas of
 2 interest each include two or more distinct sub-areas of the respective page.

1 7. The method of claim 1, wherein the step of identifying a first area
 2 of interest includes the step of identifying portions of the HTML tags of the model page.

1 8. The method of claim 1, wherein the step of identifying a first area
2 of interest is performed using a manual pointing and selecting device.

1 9. The method of claim 1, wherein the steps of selecting and
2 identifying are performed manually and wherein the remaining steps are performed
3 automatically.

1 10. The method of claim 1, wherein the second web page is retrieved
2 from a remote website over the Internet.

1 11. The method of claim 1, wherein the HTML tags include attributes
2 and attribute values.

1 12. A computer readable medium containing instructions for
2 controlling a computer system to automatically identify and extract desired content from a
3 retrieved HTML formatted web page, by automatically:
4 parsing the HTML code of a manually selected model web page to
5 determine a first string of symbols associated with a first plurality of HTML tags;
6 retrieving a second web page;
7 parsing the HTML code of the second web page to determine a second
8 string of symbols associated with HTML tags of the second page; and
9 comparing the first and second strings to determine whether the second
10 page includes a second plurality of HTML tags substantially matching the first plurality
11 of HTML tags.

1 13. The computer readable medium of claim 12, wherein the first
2 plurality of HTML tags are identified by an operator using a pointing and selection device
3 coupled to the computer system.

1 14. The computer readable medium of claim 12, wherein the second
2 web page is retrieved from a remote website over the Internet.

1 15. The computer readable medium of claim 12, further including
2 instructions for extracting a portion of the second page corresponding to the second
3 plurality of HTML tags.

1 16. The computer readable medium of claim 15, wherein the
2 instructions further control the computer system to store the extracted portion of the
3 second page in a database.

1 17. The computer readable medium of claim 15, further including
2 instructions for controlling the computer system to apply a regular expression matching
3 algorithm to the extracted portion of the second page.

1 18. The computer readable medium of claim 15, wherein the extracted
2 portion of the second page includes two or more distinct sub-areas.

1 19. The computer readable medium of claim 12, wherein the
2 instructions for comparing include instructions for applying an approximate string
3 matching algorithm to the first and second strings.

1 20. The computer readable medium of claim 12, wherein the HTML
2 tags include attributes and attribute values.

1 21. A computer system for identifying and extracting content from
2 HTML formatted web pages, the system comprising:
3 means for retrieving web pages including HTML tags, wherein a model
4 web page is retrieved;
5 means for manually identifying a first area of interest in the model page,
6 wherein the first area of interest corresponds to a first plurality of HTML tags; and
7 a processor including:
8 means for parsing a page, wherein the parsing means parses the
9 model page to determine a first string of symbols associated with the first plurality of
10 HTML tags, and wherein the parsing means thereafter parses an automatically retrieved
11 second web page to determine a second string of symbols associated with the HTML tags
12 of the second web page;

13 means for comparing the first and second strings to determine
14 whether the second string includes a second portion similar to the first portion of the first
15 string, wherein the second portion corresponds to a second area of interest in the second
16 page; and

17 means for extracting the second area of interest from the second page.

1 22. A computer implemented method of identifying and extracting
2 content from web pages formatted using a markup language, comprising the steps of:
3 selecting a model page, wherein the model page includes a plurality of
4 tokens;
5 identifying a first area of interest in the model page;
6 parsing the model page to determine a first string of symbols associated
7 with the plurality of tokens, wherein the first area of interest is identified by a first portion
8 of the first string of symbols;
9 retrieving a second web page;
10 parsing the second web page to determine a second string of symbols
11 associated with the tokens of the second web page; and
12 comparing the first and second strings to determine whether the second
13 string includes a second portion similar to the first portion of the first string, wherein the
14 second portion corresponds to a second area of interest in the second page.

1 23. The method of claim 22, further comprising the step of extracting
2 the second area of interest from the second page.

1 24. The method of claim 22, wherein the markup language is selected
2 from the group consisting of HTML, XML, WML, DHTML and HDML.

1 25. The method of claim 22, wherein the tokens include tag elements
2 and text elements.

SYSTEMS AND METHODS FOR IDENTIFYING AND EXTRACTING DATA FROM HTML PAGES

ABSTRACT OF THE DISCLOSURE

Systems and methods for analyzing HTML formatted web pages to automatically identify and extract desired information. A computer algorithm identifies and extracts different pieces of information from different web pages automatically after minimal manual setup. The algorithm automatically analyzes pages with different content if they have the same, or similar, formats. The algorithm is fast and efficient and performs the extraction process quickly in real-time. The systems and methods are useful to build databases from unstructured web information. The algorithm can be used as an agent that captures information about products, and compares prices or other characteristics. It can also be used to populate structured databases that, given the different pieces of information, can analyze products and their characteristics. And it can also be used for data mining applications looking for patterns useful for marketing analyses, or other uses.

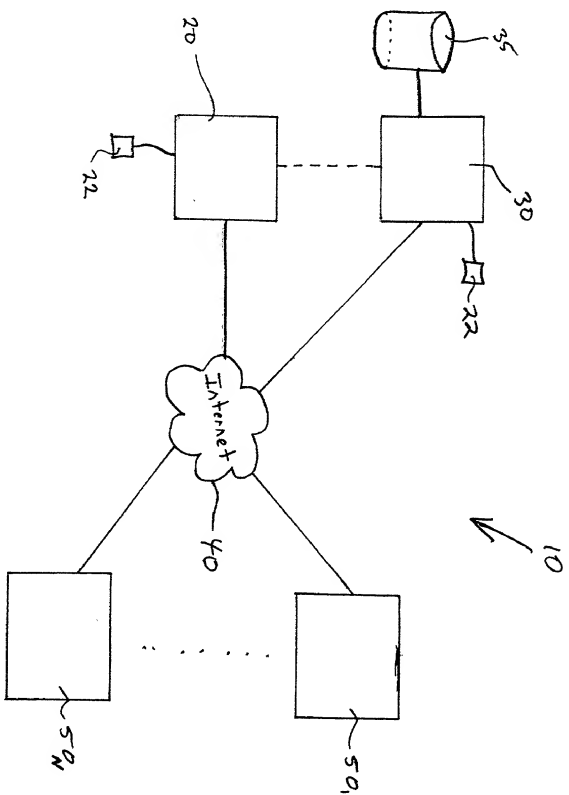


Figure 1

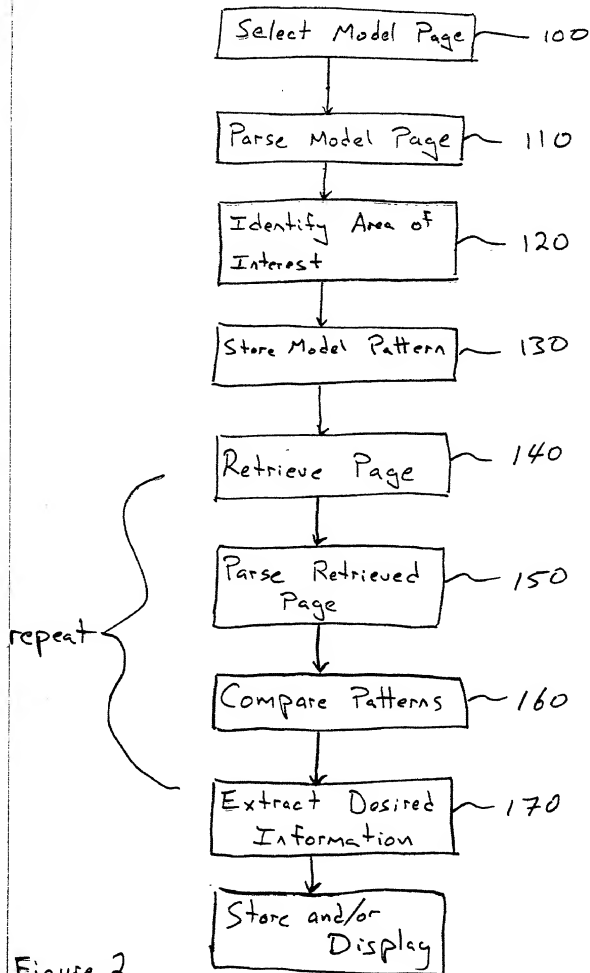


Figure 2

DECLARATION

As a below named inventor, I declare that:

My residence, post office address and citizenship are as stated below next to my name; I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural inventors are named below) of the subject matter which is claimed and for which a patent is sought on the invention entitled: **SYSTEMS AND METHODS FOR IDENTIFYING AND EXTRACTING DATA FROM HTML PAGES** the specification of which X is attached hereto or _____ was filed on _____ as Application No. _____ and was amended on _____ (if applicable).

I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above. I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56. I claim foreign priority benefits under Title 35, United States Code, Section 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed.

Prior Foreign Application(s)

Country	Application No.	Date of Filing	Priority Claimed Under 35 USC 119

I hereby claim the benefit under Title 35, United States Code § 119(e) of any United States provisional application(s) listed below:

Application No.	Filing Date

I claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56 which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

Application No.	Date of Filing	Status

Full Name of Inventor 1:	Last Name: MANBER	First Name: UDI	Middle Name or Initial:	
Residence & Citizenship:	City: Palo Alto	State/Foreign Country: California	Country of Citizenship: United States	
Post Office Address:	Post Office Address: 883 Robb Road	City: Palo Alto	State/Country: California	Postal Code: 94306
Full Name of Inventor 2:	Last Name: LU	First Name: QI	Middle Name or Initial:	
Residence & Citizenship:	City: Saratoga	State/Foreign Country: California	Country of Citizenship: United States	
Post Office Address:	Post Office Address: 20847 Russell Lane	City: Saratoga	State/Country: California	Postal Code: 95070

I further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

Signature of Inventor 1	Signature of Inventor 2
_____	_____
Udi Manber	Qi Lu
Date	Date

SF 1125961 v1

APPENDIX A

Name

cell.cc
cell.h
extr.cc
gex.cc
matcher.cc
matcher.h
pageMatcher.cc
pageMatcher.h
patElem.cc
patElem.h
patExtraction.cc
patExtraction.h
patFilter.cc
patFilter.h
patMatcher.cc
patMatcher.h
patNode.cc
patNode.h
table.cc
table.h
tablePool.cc
tablePool.h
test1.cc
tokFilter.cc
tokFilter.h

```

cell.cc

/**
 * This class represents a cell in the approximate string matching
 *   table.
 */

#include "cell.h"

yexCell::yexCell(int row, int col) {
    this->row = row; this->col = col;
    index = -1;
    setCost(-1);
    predecessor = NULL;

#ifdef TABLE_STAT
    cerr << "CELL CREATED" << endl;
    cerr << "CELL SIZE = " << sizeof(*this) << endl;
#endif

};

void yexCell::reset(int row, int col) {
    this->row = row; this->col = col;
    index = -3;
    predDirec = 0;
    setCost(-3);
    predecessor = NULL;
}

```

```

cell.h

#ifndef yexCell_h
#define yexCell_h

#include "iostream.h"

class yexCell {

private:

    yexCell::yexCell(const yexCell& ) {}
    yexCell& yexCell::operator=(const yexCell& ) {
        return *this;
    }

public:

    int cost; // this is true cost, or (simple_cost - index)
    int index;
    int row;
    int col; // logical col, set for every write to this cell

    // for error checking that write then read was same col
    yexCell* predecessor;

    int predDirec; // direction predecessor came from
    // Add these to row, col to get coords of predecessor
    //signed char predRowDelta; // 0 or -1
    //signed char predColDelta; // 0 or -1

    yexCell(int row, int col);
    //yexCell() { /*cerr << "default constr yexCell" << endl;*/};

    int getCost() {return cost;};
    void setCost(int cost) {this->cost = cost;};
    void yexCell::reset(int row, int col);

}; // class yexCell

#endif // yexCell_h

```

```

extr.cc

extern "C" {
#include "httpd.h"
#include <sys/resource.h>
#include <sys/time.h>
} // extern

#include "yut/string.h"
#include <iostream.h>
#include <fstream.h>
#include "patMatcher.h"
#include "matcher.h"

extern "C" {
    // #include "/homes/arup/src/ccmalloc-0.2.3/src/ccmalloc.h"
}

#define TIME(x, y) (int)((y).tv_sec - (x).tv_sec) * 1000000 +
    (y).tv_usec - (x).tv_usec)

void leakTest1();
void leakTestGrp(bool dontExtract = false);
void leakTestReset();
void cpuTest();

int main(int argc, char* argv[]) {
    yhpHtmlToken::tokCount = 0;
    char* infileName = NULL;
    bool genConfig = false;
    bool noExtract = false;
    yutString patDirPath = ".";
    yutString arg;
    for (int i=1; i < argc; i++) {
        arg = argv[i];
        if (arg.index("--d") == 0) {
            debugLevel_ = atoi(arg.from(2));
            pageMatcher_debugLevel_ = debugLevel_;
            cout << "ok debugLevel = " << debugLevel_ << endl;
        } else if (arg.index("-mg") == 0) { // make sure before -m
            cout << "Doing group mem leak test..." << endl;
            while (true) {
                leakTestGrp(noExtract);
                /*dbg*/ cout << "yhpHtmlToken::tokCount = " <<
yhpHtmlToken::tokCount << endl;
            }
        } else if (arg.index("-mr") == 0) { // make sure before -m
            cout << "Doing reset mem leak test..." << endl;
            leakTestReset();
            //ccmalloc_report();
            //ccmalloc_report();
        } else if (arg.index("-m") == 0) {
            cout << "Doin' single pattern mem leak test..." << endl;
            //while (true) {
                leakTest1();
            //}
        }
    }
}

```

```

        //ccmalloc_report();
        //ccmalloc_report();
        //}
    } else if (arg.index("-h") == 0) {
        cout << "USAGE: " << argv[0] << " [options] file\n";
        cout << "    Options:\n";
        cout << "        -h    -> give help info\n";
        cout << "        -m    -> perform mem leak test\n";
        cout << "        -mg   -> perform group mem leak test\n";
        cout << "        -mr   -> perform mem leak test with reset\n";
        cout << "        -n    -> no extract in mem leak test \n";
        cout << "        -g    -> generate config.gen and pattern.gen
files\n";
        cout << "        -n    -> no extract in mem leak test \n";
        cout << "        -dN   -> set debug level\n";
        cout << "        -t    -> time execution\n";
        cout << "        -pdir pat_dir_path \n";
        exit(0);

    } else if (arg.index("-g") == 0) {
        genConfig = true;
    } else if (arg.index("-t") == 0) {
        cpuTest();
    } else if (arg.index("-n") == 0) {
        cout << "Saw noExtract flag " << endl;
        noExtract = true;
    } else if (arg.index("-pdir") == 0) {
        if (argc < i) {
            cerr << "Error: Saw -pdir with no path after it." << endl;
            exit(1);
        }
        patDirPath = argv[i+1];
        i++; // skip to next arg
    } else {
        if (infileName) {
            cerr << "ERROR: saw two input file names: " << infileName <<
" and "
            << argv[i] << endl;
            exit(1);
        } else {
            infileName = argv[i];
        }
    }
}

if (!infileName) {
    infileName = "sample.htm";
}

ifstream infile(infileName, ios::in);
if (!infile) {
    cerr << "ERROR: could not open " << infileName << endl;
    return 0;
}

yexMatcher matcher;

```

```

    if (!matcher.init(patDirPath)) {
        exit(1);
    }
    matcher.setInputStream(infile);

    if (genConfig) {
        cerr << "Generating pattern.gen and config.gen..." << endl;
        matcher.writeOutConfig();
    }

    yutHash resultHash;
    matcher.startProcessing(true);
    while (matcher.nextItem(resultHash)) {
        cout << "Outputting Match results:" << endl;
        yutStringPairIterator pairs = resultHash.pairs();
        while (pairs.isValid()) {
            cout << "    Result " << pairs.key() << "=" << pairs.item() <<
endl;
                pairs.next();
            }
            resultHash.clear(); // clear hash between calls
        }
        matcher.startProcessing(false);
    } // main()

    // Reads in config just once, then extracts infinitely
    //
    void leakTestReset() {
        yexMatcher matcher;
        matcher.init(".");

        while (true) {
            ifstream infile("sample.htm", ios::in);
            if (!infile) {
                cout << "ERROR: could not open " << "sample.htm" << endl;
                return;
            }

            matcher.setInputStream(infile); // resets matcher auto'ly
            yutHash resultHash;
            matcher.startProcessing(true);
            //cout << "After init = " << yhpHtmlToken::tokCount << endl;
            //yhpHtmlToken::tokCount = 0;
            while (matcher.nextItem(resultHash)) {
                cout << "Outputting Match results:" << endl;
                yutStringPairIterator pairs = resultHash.pairs();
                while (pairs.isValid()) {
                    cout << "    Result " << pairs.key() << "=" << pairs.item() <<
endl;
                        pairs.next();
                    }
                    resultHash.clear(); // clear hash between calls
                }
                //cout << "yhpHtmlToken::tokCount 1 = " << yhpHtmlToken::tokCount
<< endl;
                matcher.startProcessing(false);
            }
        }
    }

```

```

        matcher.reset(); // not needed but match what Qi does for test
        /*dbg*/ cout << "yhpHtmlToken::tokCount = " <<
yhpHtmlToken::tokCount << endl;
    } // while true
} // leakTestReset

// actually will run on group or indiv pattern, depending on what's in
// current directory
//
void leakTestGrp(bool dontExtract = false) {

    char* inFileName = "sample.htm";
    yexMatcher matcher;
    matcher.init(".");
    ifstream infile(inFileName, ios::in);
    matcher.setInputStream(infile);

    if (dontExtract)
        return;
    yutHash resultHash;
    while (matcher.nextItem(resultHash)) {
        yutStringPairIterator pairs = resultHash.pairs();
        while (pairs.isValid()) {
            cout << "    Result " << pairs.key() << "=" << pairs.item() <<
endl;
                pairs.next();
            }
        resultHash.clear(); // clear hash between calls
    }
}

// does one patMatch of sample.htm
// if dontExtract, just creates table
//
/*
void leakTest1(bool dontExtract = false) {

    char* inFileName = NULL;
    if (!inFileName) {
        inFileName = "sample.htm";
    }

    //yhpHtmlTagToken::test();
    ifstream patternFile("pattern", ios::in);
    ifstream configFile("config", ios::in);
    ifstream filterFile("filter", ios::in);

    ifstream infile(inFileName, ios::in);
    if (!infile) {
        cerr << "ERROR: could not open " << inFileName << endl;
    }

    yexPatMatcher patMatcher;
    patMatcher.init(patternFile, configFile, filterFile);
    patMatcher.setInputStream(infile);

```

```

    if (dontExtract)
        return;

    yutHash resultHash;
    while (patMatcher.nextItem(resultHash)) {
        cout << "Outputting Match results:" << endl;
        yutStringPairIterator pairs = resultHash.pairs();
        while (pairs.isValid()) {
            cout << "    Result " << pairs.key() << "=" << pairs.item() <<
endl;
            pairs.next();
        }

        if (debugLevel_ > 1) {
            patMatcher.tableP_>dumpTable();
        }
        resultHash.clear(); // clear hash between calls
    }
} // leakTest1
*/

void leakTest1() {
    yexMatcher *matcher = NULL;
    //matcher.init(".");

    while (true) {
        ifstream infile("sample.htm", ios::in);
        if (!infile) {
            cout << "ERROR: could not open " << "sample.htm" << endl;
            return;
        }

        matcher = new yexMatcher();
        matcher->init(".");
        matcher->setInputStream(infile); // resets matcher auto'ly
        yutHash resultHash;
        matcher->startProcessing(true);
        while (matcher->nextItem(resultHash)) {
            cout << "Outputting Match results:" << endl;
            yutStringPairIterator pairs = resultHash.pairs();
            while (pairs.isValid()) {
                cout << "    Result " << pairs.key() << "=" << pairs.item() <<
endl;
                pairs.next();
            }
            resultHash.clear(); // clear hash between calls
        }
        matcher->startProcessing(false);
        matcher->reset();
        /*dbg*/ cout << "yhpHtmlToken::tokCount = " <<
yhpHtmlToken::tokCount << endl;
        delete matcher;
    } // while true
} // leakTestReset

```

```

void cpuTest() {
    struct timeval startElapsed;
    struct timeval endElapsed;
    struct rusage startRUsage;
    struct rusage endRUsage;

    yexMatcher matcher;
    matcher.init("");
    while (true) {
        ifstream infile("sample.htm", ios::in);
        if (!infile) {
            cerr << "ERROR: could not open " << "sample.htm" << endl;
            return;
        }
        // start timer
        gettimeofday(&startElapsed, NULL);
        getrusage(0, &startRUsage);

        matcher.setInputStream(infile); // resets matcher auto'ly
        yutHash resultHash;
        while (matcher.nextItem(resultHash)) {
            //cerr << "Outputting Match results:" << endl;
            yutStringPairIterator pairs = resultHash.pairs();
            /*
            while (pairs.isValid()) {
                cerr << " Result " << pairs.key() << "=" << pairs.item() <<
endl;
                pairs.next();
            }
            */
            resultHash.clear(); // clear hash between calls
        }
        // stop timer
        gettimeofday(&endElapsed, NULL);
        getrusage(0, &endRUsage);
        fprintf(stderr, "(%d) elapsed-time=%d\n", getpid(),
TIME(startElapsed, endElapsed));
        fprintf(stderr, "total-cpu-time=%d\n", TIME(startRUsage.ru_utime,
endRUsage.ru_utime) +
TIME(startRUsage.ru_stime, endRUsage.ru_stime));
        fflush(stderr);
    } // while true

} // cpuTest

```

gex.cc

```
#include "yut/string.h"
#include <iostream.h>
#include <fstream.h>
#include "patMatcher.h"
#include "pageMatcher.h"

int main(int argc, char* argv[]) {
    char* inFileName = NULL;
    yutString arg;
    for (int i=1; i < argc; i++) {
        arg = argv[i];
        if (arg.index("-d") == 0) {
            debugLevel_ = atoi(arg.from(2));
            cerr << "ok debugLevel = " << debugLevel_ << endl;
        } else {
            if (inFileName) {
                cerr << "ERROR: saw two input file names: " << inFileName <<
                " and " << argv[i] << endl;
                exit(1);
            } else {
                inFileName = argv[i];
            }
        }
    }

    if (!inFileName) {
        inFileName = "sample.htm";
    }

    ifstream groupFile("group.cfg", ios::in);
    if (!groupFile) {
        cerr << "ERROR: could not open group.cfg." << endl;
    }

    ifstream inFile(inFileName, ios::in);
    if (!inFile) {
        cerr << "ERROR: could not open " << inFileName << endl;
        return 0;
    }

    yexPageMatcher pageMatcher;

    if (!pageMatcher.init(groupFile, ".") {
        cerr << "Error in initing from pattern group file. Exiting" <<
        endl;
        exit(1);
    }
    pageMatcher.setInputStream(inFile);

    yutHash resultHash;

    while (pageMatcher.nextItem(resultHash)) {
```

```

cerr << "Outputting Match results:" << endl;
yutStringPairIterator pairs = resultHash.pairs();
while (pairs.isValid()) {
    cerr << "  Result " << pairs.key() << "=" << pairs.item() <<
endl;
    pairs.next();
}
resultHash.clear(); // clear hash between calls
}

```

```

matcher.cc

/* $Header: */

/**
General pattern matching class.

This class gets initialized on a directory and looks at what
files are present to determine if a patMatcher (single pattern
match) or pageMatcher (pattern group match) should be called.
If it sees a "group.cfg" file, assumes it is a group. Otherwise
first checks for "patdescr" file (newer pattern description file)
and if not there looks for older "pattern" and "config" formats).
**/

#include <fstream.h>
#include <iostream.h>
#include "matcher.h"
#include "patMatcher.h"
#include "pageMatcher.h"
#include "htmlTokenizer.h"
#include "htmlToken.h"
#include "htmlTagToken.h"

void yexMatcher::setInputStream(istream& istream) {
    if (isGroup) {
        pageMatcherP_>setInputStream(istream, false /*resetMatchers*/);
    } else {
        patMatcherP_>setInputStream(istream);
    }
}

bool yexMatcher::nextItem(yutHash& resultHash) {
    if (isGroup) {
        return pageMatcherP_>nextItem(resultHash);
    } else {
        return patMatcherP_>nextItem(resultHash);
    }
}

// Read in key.cfg file
bool yexMatcher::readinKeyCfg(yutString patternDir) {
    yutString keyFilePath = patternDir + "/key.cfg";
    ifstream keyFile(keyFilePath, ios::in);
    if (keyFile) {
        yhpHtmlTokenizer keyConfigTokenizer(keyFile);
        yhpHtmlToken *token;
        while ((token = keyConfigTokenizer.nextToken())) {
            if (token->type() != yhpHtmlToken::TAG_TYPE) {
                delete token;
                continue;
            } // if

            yhpHtmlTagToken *tag = (yhpHtmlTagToken *) token;
            yutString name = tag->getName();
            if (name == "key") {

```

```

        // this defines the key attributes for extracted data
        yutString keyName = tag->getAttributeValue("name");
        if (keyName.isUndefined()) {
            cerr << "key config file error, name attribute missing in Tag:"
<< tag->getContent() << endl;
            return false;
        } // fi
        keyAttrList_.addFirst(keyName);
    } else if (name == "value") {
        // this defines the value attributes for extracted data
        yutString valueName = tag->getAttributeValue("name");
        if (valueName.isUndefined()) {
            cerr << "key config file error, name attribute missing in Tag:"
<< tag->getContent() << endl;
            return false;
        } // fi
        valueAttrList_.addFirst(valueName);
    } else if (name == "total") {
        yutString valueName = tag->getAttributeValue("name");
        if (!valueName.isUndefined() && valueName != "")
            totalName_ = valueName;
    }
    delete token;
} // while
} // if (keyFile)
return true;
} // readinKeyCfg

// Returns true if OK, false if error.
// see comment at top of this file to see
// what files this checks for.
bool yexMatcher::init(yutString patternDir) {
    // we first look to see if there is a key.cfg file that defines the
    key/value attributes

    if (!readinKeyCfg(patternDir)) {
        return false;
    }

    yutString path = patternDir + "/group.cfg";
    // first look for group file
    ifstream groupFile(path, ios::in);
    if (groupFile) {
        isGroup = true;
        pageMatcherP_ = new yexPageMatcher;
        return pageMatcherP_->init(groupFile, patternDir);
    } else {
        isGroup = false;
        patMatcherP_ = new yexPatMatcher;
        return patMatcherP_->init(patternDir);
    }

    return false;
} // init

// process token, return true if there is a match of a pattern

```

```

//
bool yexMatcher::processToken(yhpHtmlToken* tokenP) {
    if (isGroup) {
        return pageMatcherP_>processToken(tokenP);
    } else {
        return patMatcherP_>processToken(tokenP);
    }
} // processToken

// Call this after determining there is a match (processToken
// returns true). Extracted vals added hash
void yexMatcher::getMatchResult(yutHash& hash) {
    if (isGroup) {
        pageMatcherP_>getMatchResult(hash);
    } else {
        patMatcherP_>getMatchResult(hash);
    }
}

// used usually when patdescr is read in to write out
// pattern and config files
//
bool yexMatcher::writeOutConfig() {
    if (!isGroup) {
        return patMatcherP_>writeOutConfig("pattern.gen", "config.gen");
    }
    return false;
} // writeOutConfig

// constr
yexMatcher::yexMatcher() {
    isGroup = false;
    pageMatcherP_ = NULL;
    patMatcherP_ = NULL;
    totalName_ = "total"; // this is the default name for the total
    field for auto-summation
}
// destr
yexMatcher::~yexMatcher() {
    if (isGroup) {
        if (pageMatcherP_) delete pageMatcherP_;
    } else {
        if (patMatcherP_) delete patMatcherP_;
    }
    pageMatcherP_ = NULL;
    patMatcherP_ = NULL;
}

void yexMatcher::reset() {
    if (isGroup)
        pageMatcherP_>reset();
    else
        patMatcherP_>reset();
}

void yexMatcher::startProcessing(bool start) {

```

```
if (isGroup) {  
    pageMatcherP_>startProcessing(start);  
} else {  
    patMatcherP_>startProcessing(start);  
}  
}
```

004260-02454960

matcher.h

```
/* $Header: */

#ifndef yexMatcher_h
#define yexMatcher_h

#include "yex/patMatcher.h"
#include "yex/pageMatcher.h"
#include "yut/string.h"

class yhpHtmlToken;
class yutHash;
class yutString;

class yexMatcher {
private:
    bool isGroup;
    yexPatMatcher* patMatcherP_;
    yexPageMatcher* pageMatcherP_;
    yutStringList keyAttrList_; // a list of names for those extracted
    attributes that are used as keys
    yutStringList valueAttrList_; // a list of names for those extracted
    attributes that are used as values
    yutString totalName_; // the name of the total field when used
    during automatic summation process

public:
    yexMatcher();
    ~yexMatcher();
    bool init(yutString patternDir);
    bool processToken(yhpHtmlToken* tokenP);
    void getMatchResult(yutHash& hash);
    bool writeOutConfig();
    void setInputStream(istream& istream);
    bool nextItem(yutHash& resultHash);
    void reset();
    inline const yutStringList &getKeyAttrList() const { return
keyAttrList_; }
    inline const yutStringList &getValueAttrList() const { return
valueAttrList_; }
    inline const yutString &getTotalName() const { return totalName_; }
    bool yexMatcher::readinKeyCfg(yutString patternDir);

    void startProcessing(bool start);
}; // class yexMatcher

#endif // yexMatcher_h
```

pageMatcher.cc

/**

* Matches an html page, which is represented as a DAG (directed,
acyclical graph)

Rules of the game:

- Definitions:
("multi" means mutli-match pattern. "single" means single-match pattern. An "intertwine" is a parGroup of with at least one multi directly as child. Name refers to fact that if you have 2 or more multi;s then they "intertwine" matches.
- A multi must always appear directly within a ParGroup, the parGroup must contain only leaf pats, with at least one single in it to define the exit condition, the parGroup is implicitly intertwined, and it must be directly preceded by a single.
- There can be a par that is not an intertwine - it is always a branch-out (eventually) to all singles.
- We declare multi's explicitly within the parGroup (since same pattern may be the single above it).

Resulting Graph structure:

- fanout and fanin nodes never contain a pat
- pat node has at most one child
- fanin node has at most one child. fanin != fanout ever.

*/

extern int strmToker_debug;

```
// stdlib is for atoi()
#include <stdlib.h>
#include <fstream.h>
#include <iostream.h>
#include <vector.h>
#include "htmlTagToken.h"
#include "patMatcher.h"
#include "patNode.h"
#include "pageMatcher.h"
```

int pageMatcher_debugLevel_ = 0;

```
// constructor
yexPageMatcher::yexPageMatcher() {
    htmlTokerP_ = new yhpHtmlToker();
    htmlTokerP_->outputWhitespaceTextToks(false); // wasfalse
    htmlTokerP_->setFixMissingCloseQuote(true);
    done_ = false;
    inputStreamSet = false;
}

// destructor
yexPageMatcher::~yexPageMatcher() {
```

```

    if (htmlTokenizerP_) delete htmlTokenizerP_;
    htmlTokenizerP_ = NULL;

    // delete all graph nodes
    for (int i = 0; i < (int) nodeList.size(); i++) {
        if (nodeList_[i]) delete nodeList_[i];
        nodeList_[i] = NULL;
    }
    // Deallocate the patMatchers from the patMatcherMap, since it
    // has unique list of them
    OTC_Iterator<yexPatMatcher*> it = patMatcherMap_.items();
    for(it.reset(); it.isValid(); it.next()) {
        if (it.item()) delete it.item();
    }
}

// Returns true if OK, false if error.
bool yexPageMatcher::init(istream& groupStream, yutString patternDir) {

    // Maps from the name of the pat group to the tag that holds it.
    // used as temp d/s when building up the pattern graph
    //
    patGroupMapT patGroupMap; // maps group name -> tag holding group
    //patMatcherMapT patMatcherMap; // maps pat name -> patMatcher
    pointer

    patternDirPath_ = patternDir;
    if (!readInConfig(groupStream, patGroupMap, patMatcherMap_) {
        return false;
    }
    graphHead_ = buildPatGraph("top", patGroupMap, patMatcherMap_);

    // Deallocate the tags from the patGroupMap, since done with them.
    OTC_Iterator<yhpHtmlTagToken*> it = patGroupMap.items();
    for(it.reset(); it.isValid(); it.next()) {
        if (it.item()) delete it.item();
    }

    if (pageMatcher_debugLevel_ > 1) {
        dumpGraph();
    }

    if (!graphHead_) {
        return false;
    }

    if (!checkGraph()) {
        return false;
    }

    // Fill in initial matchList_
    propagateFromNode(graphHead_);

    if (pageMatcher_debugLevel_ > 2) {
        cerr << "Matche nodes intially:" << endl;
    }
}

```

```

    cerr << "matchList_.size() = " << matchList_.size() << endl;

    for (unsigned int i = 0; i < matchList_.size(); i++) {
        cerr << "match node:" << matchList_[i]->name_ << endl;
    }

    return (graphHead_ != NULL); // non-null graph means success
} // init

// process token, return true if there is a match of a pattern
//
bool yexPageMatcher::processToken(yhpHtmlToken* tokenP) {
    if (debugLevel_ > 3) {
        cerr << "page token is " ;
        if (tokenP) tokenP->describeToken();
    }

    for (unsigned int i = 0; i < matchList_.size(); i++) {
        if (i > 0 && tokenP) { // don't increment count copy for 1st use --
            it's implicit
            tokenP->incrRefCount();
        }
    }

    for (unsigned int i = 0; i < matchList_.size(); i++) {
        yexPatNode* candidate = matchList_[i];

        if (debugLevel_ > 3) {
            //cerr << candidate->name_ << " ";
        }
        //cout << "process token" << endl;
        if (candidate->getLeafPatMatcher()->processToken(tokenP)) {
            //cout << "-----MATCHED-----" << endl;

            for(unsigned int j = i + 1; j < matchList_.size(); j++) {
                yexPatNode *node = matchList_[j];
                node->getLeafPatMatcher()->processToken(tokenP);
            }

            matchedPat_ = candidate;
            if (candidate->isMultiMatch_) {
                // multi-matcher matched.
                if (pageMatcher_debugLevel_ > 2) {
                    cerr << "Multi match matched: " << candidate->name_ << endl;
                }
            } else {
                if (pageMatcher_debugLevel_ > 2) {
                    cerr << "Single match matched: " << candidate->name_ << endl;
                }
            }
            // single match -- move down graph to prepare for
            // next call to processToken()
            if (candidate->children_.size() < 1) {
                done_ = true; // nowhere to propagate to
            }
        }
    }
}

```

```

        return true;
    }
    OTCLIB_ASSERT(candidate->children_.size() == 1); // pattern nodes
    have at most 1
    clearMatchList();
    propagateFromNode(candidate->children_[0]);
    if (pageMatcher_debugLevel_ > 2) {
        cerr << "Matched nodes after propagate: " << matchList_.size()
    }
    << endl;
    for (unsigned int j = 0; j < matchList_.size(); j++) {
        cerr << "    " << matchList_[j]->name_ << endl;
    }
    }
    // Reset all after any match.
    //for (unsigned int j = 0; j < matchList_.size(); j++) {
    //    matchList_[j]->getLeafPatMatcher()->reset();
    //}

    // If a single match was at end of graph we are done. Indicated
    by
    // fact that match list is empty
    if (matchList_.size() < 1) {
        done_ = true;
    }
    return true;
}
return false;
}

// Call this after determining there is a match (processToken
// returns true). Extracted vals added hash
void yexPageMatcher::getMatchResult(yutHash& hash) {
    matchedPat_->getLeafPatMatcher()->getMatchResult(hash);
    // Reset all after any match.
    for (unsigned int j = 0; j < matchList_.size(); j++) {
        matchList_[j]->getLeafPatMatcher()->reset();
    }
}

void yexPageMatcher::clearMatchList() {
    //
    while (!matchList_.empty()) {
        matchList_.pop_back();
    }
}

// Propagate from the given pattern node in the graph
// "downward" (thru children), stopping when you get
// to leaf nodes (nodes with pattern), and add
// leaf nodes to matchList_. If called on a leaf node
// directly, will add that node only and stop.
//
void yexPageMatcher::propagateFromNode(yexPatNode* pNode) {

```

```

        if (pageMatcher_debugLevel_ > 2) {
            cerr << "Propagate: called on node: " << pnode->name_ << endl;
        }
        if (pnode->getLeafPatMatcher()) {
            if (pageMatcher_debugLevel_ > 2) {
                cerr << "Propagate: adding node that has a pat: " << pnode-
>name_ << endl;
            }

            matchList_.push_back(pnode); // add to matchList
            return;
        }
        for (unsigned int i = 0; i < pnode->children_.size(); i++) {
            // call recursively on children
            OTCLIB_ASSERT(pnode->children_[i]); // why have null children?
            propagateFromNode(pnode->children_[i]);
        }
    }

    void yexPageMatcher::dumpGraph() {
        for (int i = 0; i < (int) nodeList_.size(); i++) {
            cerr << "node " << nodeList_[i]->name_ << " has " << nodeList_[i]-
>children_.size() << " children: " << endl;
            for (int j = 0; j < (int) nodeList_[i]->children_.size(); j++) {
                cerr << "    " << nodeList_[i]->children_[j]->name_ << endl;
            }
        }
    } // dumpGraph

    // Reads in pattern group config file.
    // Fills in leafPatNames_ with leaf names from config file,
    // and fills in patGroupMap. Returns true if OK,
    // false if error.
    //
    bool yexPageMatcher::readInConfig(istream& configStream,
                                      patGroupMapT& patGroupMap,
                                      patMatcherMapT& patMatcherMap) {

        yhpHtmlToken* tokenP;

        yhpHtmlToker configToker(configStream);
        while ((tokenP = configToker.nextToken())) {

            if (tokenP->type() != yhpHtmlToken::TAG_TYPE) {
                delete tokenP; // delete config file token
                continue;
            }

            yhpHtmlTagToken* tagP = (yhpHtmlTagToken*) tokenP;

            /*delete
            delete tokenP; // delete config file token
            yhpHtmlTagToken tag;

```

```

        tag = *tagP;
        */
#ifdef _HASH_ENABLED_

        yutString name = tagP->getName();

#else

        int tagType = tagP->getType();

#endif

#ifdef _HASH_ENABLED_

        if (name == "leafpats") {

#else

        if (tagType == yhpHtmlTagToken::hash("leafpats")) {

#endif

        yutString pats = tagP->getAttributeValue("pats");
        pats.split(leafPatNames_, ",");
        delete tagP;

#ifdef _HASH_ENABLED_

        } else if (name == "seqgroup" || name == "pargroup" ) {

#else

        } else if (tagType == yhpHtmlTagToken::hash("seqgroup") ||
            tagType == yhpHtmlTagToken::hash("pargroup")) {

#endif

        yutString groupName = tagP->getAttributeValue("name");
        patGroupMap.add(groupName, tagP); // add to map for later
lookup
        // will delete tags when done, in init()

#ifdef _HASH_ENABLED_

        } else if (name == "config") {

#else

        } else if (tagType == yhpHtmlTagToken::hash("config")) {

#endif

        yutString level = tagP->getAttributeValue("debuglevel");
        if (level != "") {
            debugLevel_ = atoi(level);
            cerr << "set debugLevel from config tag to " << debugLevel_
<< endl;

```

```

    }
    level = tagP->getAttributeValue("htdebuglevel");
    if (level != "") {
        strmTokenizer_debug = atoi(level);
        cerr << "set ht_debugLevel from config tag to " <<
            strmTokenizer_debug << endl;
    }
    delete tagP;
}
// patGroupMap
} // while

// Now take pat leaf names and read in their config files
yutString patternSubdir;
yutStringIterator it = leafPatNames_.items();
for(it.reset(); it.isValid(); it.next()) {
    patternSubdir = patternDirPath_ + "/" + it.item();
    yexPatMatcher* patMatcher = new yexPatMatcher; // set before
    init() since affects lookahead

    patMatcher->patName_ = it.item(); // name patMatcher.

    if (!patMatcher->init(patternSubdir)) { // uses patdescr if
exists
        return false;
    }
    patMatcherMap.add(it.item(), patMatcher);
}
return true;
} // readInConfig

// Build the pattern graph rooted at pattern or patGroup "name"
// and return the head(root), or NULL if error. Call this with
// "top" as name to get the whole enchilada. Calls itself
// recursively on sub-groups, recursion termination condition is
// when 'name' names a leaf pat.
//
yexPatNode* yexPageMatcher::buildPatGraph(yutString name, patGroupMapT&
patGroupMap,
                                     patMatcherMapT& patMatcherMap) {

    // for terryO:
    //cerr << "called buidGraph on " << name << endl;
    yexPatNode* pNode = NULL; // node to return

    bool isMultiPat = true; // default to yes
    int index;
    // first see if it names a leaf pat
    // check for patname:multi too
    if ((index = name.index(":single")) > 0) {
        isMultiPat = false;
        name = name.before(index);
    }

    if (patMatcherMap.contains(name)) {
        pNode = new yexPatNode;
    }

```

```

    pnode->isMultiMatch_ = isMultiPat;
    pnode->name_ = name;
    nodeList_.push_back(pnode); // add to nodelist
    pnode->setLeafPatMatcher(patMatcherMap.item(name));
    return pnode;

    // check if a pattern group by that name exists
} else if (patGroupMap.contains(name)) {
    yhpHtmlTagToken* tagP;
    tagP = patGroupMap.item(name);

    yutString elemsStr = tagP->getAttributeValue("elems");
    yutStringList elems;
    elemsStr.split(elems, ",");

#ifdef _HASH_ENABLED_

    if (tagP->getName() == "seqgroup") {

#else

    if (tagP->getType() == yhpHtmlTagToken::hash("seqgroup")) {

#endif

        // sequential group. For each elem of the group
        // (which may be a graph), we take the exit node
        // and link it to the next elem.

        yexPatNode* lastChild = NULL;
        yutStringIterator it = elems.items();

        for(it.reset(); it.isValid(); it.next()) {
            //cerr << "seq child name " << it.item() << endl;
            // call recursively on cur child
            yexPatNode* curChild =
                buildPatGraph(it.item(), patGroupMap, patMatcherMap);
            if (curChild == NULL) {
                return NULL; // if fatal error, we die too
            }
            if (pnode == NULL) {
                pnode = curChild; // point to first child as returned node
            } else {
                // 2nd or after child
                OTCLIB_ASSERT(lastChild);
                lastChild->exitNode()->addChild(curChild);
            }
            lastChild = curChild;
        }
        return pnode;

#ifdef _HASH_ENABLED_

    } else if (tagP->getName() == "pargroup") {

#else

    } else

```

```

    } else if (tagP->getType() == yhpHtmlTagToken::hash("pargroup")) {
#endif

        // parallel group. A fanout node is created,
        // whose children are the children of this parGroup,
        // and then a fanin node that will be the child of all
        // of them. pnode is fanout node
        pnode = new yexPatNode;
        pnode->name_ = name;
        nodeList_.push_back(pnode); // add to nodeList

        yexPatNode* tailNode = new yexPatNode;
        tailNode->name_ = "\\\" + name;
        nodeList_.push_back(tailNode); // add to nodeList

        yutStringIterator it = elems.items();

        for(it.reset(); it.isValid(); it.next()) {
            //cerr << "par child name " << it.item() << endl;
            // call recursively on cur child
            yexPatNode* curChild =
                buildPatGraph(it.item(), patGroupMap, patMatcherMap);
            if (curChild == NULL) {
                return NULL; // if fatal error, we die too
            }
            pnode->addChild(curChild);

            // Since curChild may be a graph itself, get exitNode
            // and link that to tailNode.
            curChild->exitNode()->addChild(tailNode);
        }
        return pnode;
    } else {
        // Neither pattern nor group with name -- error
        cerr << "ERROR: in building pattern group map. This pattern or
group \nwas not found. Name: " << name << endl;
        return NULL;
    }

    return pnode;
} // buildPatGraph

// Check the pattern graph for "rules" and return true if OK,
// also will set the flag for a fanout that is an
// "intertwine" group (see rules at top this file for definition)
//
bool yexPageMatcher::checkGraph() {

    for (int i = 0; i < (int) nodeList_.size(); i++) {
        yexPatNode* pnode = nodeList_[i];
        if (pnode->isMultiMatch_) {
            OTCLIB_ASSERT(pnode->getLeafPatMatcher()); // if multi must have
pattern

```

```

    // Check for case of multi being at end of graph, i.e. keeps
    // matching until end of stream. If at end, then rest of
    // multi rules don't apply. End node means as we go down graph
there
    // are no patterns (after the multi), and the graph doesn't
branch.
    //

yexPatNode* tnode = pnode; // temp node to ride down graph
bool isEndNode = true; // assume it is until proven otherwise
while (tnode) {

    if (tnode->children_.size() > 1) {
        isEndNode = false;
        break;
    }

    // 2nd part of check allows mutli to have a pattern but
    // not its descendants
    if (tnode->getLeafPatMatcher() && tnode != pnode) {
        isEndNode = false;
        break;
    }
    if (tnode->children_.size() == 0) {
        break;
    }
    tnode = tnode->children_[0]; // follow only child
}

if (isEndNode) {
    continue;
}

if (pnode->getParent()) {
}

if (!pnode->getParent() || pnode->getParent()->children_.size() <
2) {
    cerr << "Error in group pattern config. Pattern " << pnode->name_
<< endl;
    cerr << "is a multi match, but does not have a parent with
multiple children."
<< endl
<< " (a multi must be directly in a parGroup with at least one
single-match)"
<< endl;
    cerr << "\n\nPattern graph:\n\n";
    dumpGraph();
    return false;
}

// make sure parent has a single match
yexPatNode* parent = pnode->getParent();

```

```

OTCLIB_ASSERT(parent); // test above

bool foundSingle = false;

for (unsigned int j = 0; j < parent->children_.size(); j++) {
    if (!parent->children_[j]->isMultiMatch_) {
        foundSingle = true;
        break;
    }
}
if (!foundSingle) {
    cerr << "Error in group pattern config. Pattern " << pnode->name_
    << endl;
    cerr << "is a multi-match, but does not have a single-match
    sibling."
    << endl;
    << "(a multi must be directly in a parGroup with at least one
    single-match)"
    << endl;
    cerr << "\n\nPattern graph:\n\n";
    dumpGraph();
    return false;
}
// Now make sure the parGroup is right after a single pattern
if (!parent->getParent() || !parent->getParent()-
>getLeafPatMatcher()
    || parent->getParent()->isMultiMatch_) {
    cerr << "Error in group pattern config. Pattern " << pnode->name_
    << endl;
    cerr << "is a multi-match, but does not follow directly a single-
    match."
    << endl;
    cerr << "\n\nPattern graph:\n\n";
    dumpGraph();
    return false;
}

// it's legit -- set intertwine bit on the parent
parent->isIntertwine_ = true;
}
}

return true;
} // checkGraph

// Sets a new input stream. If you don't want to reset the
// state of the patMatchers (usually you do), then pass
resetMatcher=false
//
void yexPageMatcher::setInputStream(istream& istream, bool resetMatcher)
{
    // When you setInputStream the 2nd and subsequent times, the htmlToker
    // is reset automatically
    //

```

```

    if (inputStreamSet) { // if this is 2nd or more time you've reset
                          // input stream (ie you are resetting it).
        //cout << "New tok" << endl;
        if (htmlTokenizerP_) delete htmlTokenizerP_;
        htmlTokenizerP_ = new yhpHtmlTokenizer();
        htmlTokenizerP_>outputWhitespaceTextToks(false);
        htmlTokenizerP_>setFixMissingCloseQuote(true);
        reset();
    }
    htmlTokenizerP_>setInputStream(istream);
    inputStreamSet = true;
    if (resetMatcher) {
        reset();
    }
};

// Resets state of the patMatchers. Do not need to call this
// if you change inputStream since handled in that routine.
//
void yexPageMatcher::reset() {
    done_ = false;

    for (int i = 0; i < (int) nodeList_.size(); i++) {
        if (nodeList_[i]>getLeafPatMatcher()) {
            //cout << "RESET " << i << endl;
            nodeList_[i]>getLeafPatMatcher()->reset();
        }
    }
}

// returns true and puts next item (set of name-value pairs) into hash
// if match, else returns false indicating at EOF and no more matches
// (or could also be that had single match pattern at end of
// graph that matched so we are done.
bool yexPageMatcher::nextItem(yutHash& resultHash) {
    if (debugLevel_ > 0) {
        cerr << "begin yexPageMatcher::nextItem " << endl;
    }

    yhpHtmlToken* tokenP;

    while (true) {
        if (done_) {
            if (debugLevel_ > 0) {
                cerr << "returning from yexPageMatcher::nextItem since done_"
<< endl;
            }
            return false;
        }
        tokenP = htmlTokenizerP_>nextToken();

        if (tokenP && pageMatcher_debugLevel_ > 4) {
            //tokenP->describeToken();
            cerr << "^";
        }
    }
}

```

```

    if (processToken(tokenP)) {
        getMatchResult(resultHash);

        //cout << "TOKEN = " << tokenP->toWellFormedString() << endl;
        //tokenP->decrRefCount();

        return true;
    }

    if (tokenP == NULL) {
        /* need to incr tokCount_ for this to work
        if (tokCount_ < 1) {
            cerr << "**** ERROR (yexPageMatcher::nextItem): read zero
tokens" << endl;
        }
        */
        done_ = true;
        if (debugLevel_ > 0) {
            cerr << "returning from yexPageMatcher::nextItem since null
token" << endl;
        }
        return false;
    }
} // nextItem()

void yexPageMatcher::startProcessing(bool start) {
    OTC_Iterator<yexPatMatcher*> it = patMatcherMap_.items();
    for(it.reset(); it.isValid(); it.next()) {
        if (it.item()) it.item()->startProcessing(start);
    }
}

```

```

pageMatcher.h

/* $Header: */

// Matches an html page, which is represented as a DAG (directed,
// acyclical graph)

#ifdef yexPageMatcher_h
#define yexPageMatcher_h
#include "iostream.h"
#include <vector.h>
#include "yut/string.h"
#include "yhp/htmlTagToken.h"

typedef OTC_HMap<yutString, yhpHtmlTagToken*> patGroupMapT; // grp name
-> tag of group
typedef OTC_HMap<yutString, yexPatMatcher*> patMatcherMapT; // pat name
-> patMatcher
class yexPatNode;

extern int pageMatcher_debugLevel_;

class yexPageMatcher {
private:
    yhpHtmlTokenizer* htmlTokenizer_;
    bool done_; // no more matches can be given. Either: at EOF and no
more matches // (or could also be that had single match pattern at
end of // graph that matched so we are done.
    int tokCount_; // num html tokens we have seen with this groups
// graph that matched so we are done.
    yexPatNode* graphHead_;
    vector<yexPatNode*> nodeList_; // list of all nodes in graph. Easy
way to visit all
    vector<yexPatNode*> matchList_; // list of pattern nodes currently
being matched. // if only one in list then it is a
single-match.
    patMatcherMapT patMatcherMap_; // maps pat name -> patMatcher
pointer

    yexPatNode* matchedPat_;
    yutStringList leafPatNames_;
    yutString patternDirPath_; // pathname of pattern directory that
has subdirs (e.g. pl) for patterns
    bool checkGraph();
    bool inputStreamSet; // true if has ever been set

public:
    yexPageMatcher();
    ~yexPageMatcher();

    bool readInConfig(istream& configStream);
    bool yexPageMatcher::readInConfig(istream& configStream,
patGroupMapT& patGroupMap,

```

```

                                patMatcherMapT& patMatcherMap);
    yexPatNode* yexPageMatcher::buildPatGraph(yutString name,
patGroupMapT& patGroupMap,                                patMatcherMapT& patMatcherMap);
    bool init(istream& groupStream, yutString patternDir);
    void yexPageMatcher::dumpGraph();
    void yexPageMatcher::propagateFromNode(yexPatNode* pNode);
    void yexPageMatcher::getMatchResult(yutHash& hash);
    bool yexPageMatcher::processToken(yhpHtmlToken* tokenP);
    void yexPageMatcher::setInputStream(istream& istream, bool
resetMatcher = true);
    void yexPageMatcher::reset();
    bool yexPageMatcher::nextItem(yutHash& resultHash);
    void yexPageMatcher::clearMatchList();

    void startProcessing(bool start);

}; // class yexPageMatcher

#endif // yexPageMatcher_h

```

patElem.cc

```
/**
 * A pattern element is one token of a pattern, and the associated
 * filters and extractors
 */

#include <vector.h>
#include <OTC/debug/assert.h>
#include "htmlToken.h"
#include "htmlTagToken.h"
#include "patElem.h"
#include "patFilter.h"
#include "patExtraction.h"

int yexPatElem_debugLevel = 0;

// constructor
yexPatElem::yexPatElem(yhpHtmlToken* token) :
    tokenP_(token)
{
#ifdef TABLE_STAT
    cerr << "PATTERN ELEMENT CREATED" << endl;
    cerr << "PATTERN ELEMENT SIZE = " << sizeof(*this) << endl;
#endif

    directionCost[yexTable_HORIZ] = DEFAULT_RIGHT_SCORE;
    directionCost[yexTable_VERT] = DEFAULT_DOWN_SCORE;
    directionCost[yexTable_DIAG] = DEFAULT_DIAGONAL_SCORE;
} // constructor

// destructor
yexPatElem::~yexPatElem() {
    // delete the filters_ vector
    for (int i = 0; i < (int) filters_.size(); i++) {
        if (filters_[i]) delete filters_[i];
        filters_[i] = NULL;
    }
    // delete the extractions_ vector
    for (int i = 0; i < (int) extractions_.size(); i++) {
        if (extractions_[i]) delete extractions_[i];
        extractions_[i] = NULL;
    }
    // delete the token for this pattern element
    if (tokenP_) {
        //tokenP_-->decrRefCount(); // will delete if last reference attn
        delete tokenP_; // safe to delete since not from data stream
        tokenP_ = NULL;
    }
}
```

```

void errorMsg(yutString msg) {
    cerr << "ERROR: " << msg << endl;
}

// Applies a tag from config or patdescr file to this patElem
//
void yexPatElem::applyConfigTag(yhpHtmlTagToken& tag) {
    yutString name = tag.getName();

#ifdef _HASH_ENABLED_
    if (name == "score") {
#else
        int tagType = tag.getType();
        if (tagType == yhpHtmlTagToken::hash("score")) {
#endif

        yutString valueStr = tag.getAttributeValue("value");
        if (valueStr.isUndefined()) {
            errorMsg("missing value in score tag:" + tag.getContent());
            return;
        }
        yutString op = tag.getAttributeValue("op");
        if (op == "") {
            errorMsg("missing op in score. Tag:" + tag.getContent());
            return;
        }

        int value = atoi(valueStr);
        if (op == "right") {
            directionCost[yexTable_HORIZ] = value;
        } else if (op == "down") {
            directionCost[yexTable_VERT] = value;
        } else if (op == "diagonal") {
            directionCost[yexTable_DIAG] = value;
        }

#ifdef _HASH_ENABLED_
    } else if (name == "filter") {
#else
        if (tagType == yhpHtmlTagToken::hash("filter")) {
#endif

        yexPatFilter* pf = new yexPatFilter();
        pf->attribute_ = tag.getAttributeValue("attribute");
        pf->value_ = tag.getAttributeValue("value");
    }
}

```

```

        pf->setOp(tag.getAttributeValue("op"));
        addFilter(pf);
        return;

#ifdef _HASH_ENABLED_

    } else if (name == "extraction") {

#else

    } else if (tagType == yhpHtmlTagToken::hash("extraction")) {

#endif

        yexPatExtraction* ex = new yexPatExtraction();

        ex->from_ = tag.getAttributeValue("from");
        ex->to_ = tag.getAttributeValue("to");
        ex->op_ = tag.getAttributeValue("op");
        ex->default_ = tag.getAttributeValue("default");
        ex->removeSpace_ = (tag.getAttributeValue("removespace") ==
("yes"));

        addExtraction(ex);
        return;
    }
} // applyConfigTag

// returns true iff this patElem matched the passed token
// (applying filters as needed).
//
bool yexPatElem::matchesToken(yhpHtmlToken* token) {

    //cerr << "compared token is " << endl;
    //token->describeToken();

    if (token->type() != tokenP->type()) {
        //cerr << "type mismatch false" << endl;
        return false;
    }

    if (token->type() == yhpHtmlToken::TAG_TYPE) {
        yhpHtmlTagToken* tagToken = (yhpHtmlTagToken*) token;
        yhpHtmlTagToken* tagTokenP_ = (yhpHtmlTagToken*) tokenP_;

#ifdef _HASH_ENABLED_

        if (!{tagToken->getName() == tagTokenP_->getName()} &&

#else

        if (!{tagToken->getType() == tagTokenP_->getType()} &&

#endif

        tagToken->isClosingTag() == tagTokenP_->isClosingTag())) {
        return false;
    }

```

```

    }
}
//return true; // temporary xxx

// if there are no filters this returns true attn
return passesFilters(token);

} // matchesToken

// Returns true if the passed token passes all the
// filters (usu. 1) of this patElem
//
bool yexPatElem::passesFilters(yhpHtmlToken* token) {
    for (int i = 0; i < (int) filters_.size(); i++) {
        //cerr << endl << "matchesToken this is " << tokenP_
        >getContent();

        if (!filters_[i]->passesFilter(token)) {
            if (yexPatFilter_debugLevel > 0) cerr << "filter returned
            false" << endl;

            return false;
        } else {
            if (yexPatFilter_debugLevel > 0) cerr << "filter returned
            true" << endl;
        }
    }
    return true;
} // passesFilters

// Extract from all extractions of this patElem
//
void yexPatElem::extract(yutHash& hash, yhpHtmlToken* token) {
    for (int i = 0; i < (int) extractions_.size(); i++) {
        OTCLIB_ASSERT(extractions_[i]);
        extractions_[i]->extract(hash, token);
    }
} // extract

```

patElem.h

```
// Stores an element of a pattern of htmlTokens to be matched

#ifndef yexPatElem_h
#define yexPatElem_h

#include "iostream.h"
#include <vector.h>

class yhpHtmlToken;
class yhpHtmlTagToken;
class yexPatFilter;
class yexPatExtraction;
class yutHash;

class yexPatElem {

private:

public:
    const int DEFAULT_DOWN_SCORE = 1;
    const int DEFAULT_RIGHT_SCORE = 1;
    const int DEFAULT_DIAGONAL_SCORE = 100000; // we never change

    vector<yexPatFilter*> filters_;
    vector<yexPatExtraction*> extractions_;

    vector<yhpHtmlTagToken*> configTags_;

    yhpHtmlToken* tokenP_;
    enum yexTable_directionT {yexTable_HORIZ, yexTable_VERT,
yexTable_DIAG};

    int directionCost[3];

    yexPatElem(yhpHtmlToken* token);
    ~yexPatElem();
    void applyConfigTag(yhpHtmlTagToken& tag);
    void addFilter(yexPatFilter* filter) {
        filters_.push_back(filter);
    };
    void addExtraction(yexPatExtraction* extraction) {
        //cerr << "*** addExtraction called " << this << " size="
        //      << extractions_.size() << endl;
        extractions_.push_back(extraction);};
    bool matchesToken(yhpHtmlToken* token);
    bool passesFilters(yhpHtmlToken* token);
    void extract(yutHash& hash, yhpHtmlToken* token);

}; // class yexPatElem

typedef vector<yexPatElem*> yexPatElem_Vect;

#endif // yexPatElem_h
```

patExtraction.cc

```
/**
 * A pattern extraction is associated with a patElem and gives info
 * needed to extract data from a tag
 **
 */

#include "patExtraction.h"
#include "htmlToken.h"
#include "htmlTagToken.h"
#include "yut/hash.h"
#include <OTC/Regex.hh>

int yexPatExtraction_debugLevel = 0;

// constructor
yexPatExtraction::yexPatExtraction() {
    default_ = "";
} // constructor

void yexPatExtraction::extract(yutHash& hash, yhpHtmlToken* token) {
    if (yexPatExtraction_debugLevel > 1) {
        cerr << "extracting from " << from_ << " to " << to_ << ". Tok is:
" << endl;
        token->describeToken();
    }

    yutString extractedValue = OTC_String::undefinedString();
    // no matter what kind of tag, return content for $this
    if (from_ == "$this") {
        extractedValue = token->getContent();
    } else if (token->type() == yhpHtmlToken::TAG_TYPE) {
        extractedValue = ((yhpHtmlTagToken*) token)-
>getAttributeValue(from_);
    } else if (token->type() == yhpHtmlToken::TEXT_TYPE && from_
=="text" ||
        token->type() == yhpHtmlToken::COMMENT_TYPE && from_
=="comment") {
        extractedValue = token->getContent();
    }

    if (extractedValue.isUndefined()) {
        if (yexPatExtraction_debugLevel > 1) cerr << "Empty extracted value
seen, nothing extracted" << endl;
        return;
    }

    if (op_.index("regex:") == 0) { // if op starts with "regex:"
        yutString patStr = op_.from(6);
        OTC_Regex pattern(patStr);
        if (pattern.isValid()) {
            if (pattern.match(extractedValue)) {

                // Get stuff in first parens only

```

```

        extractedValue = extractedValue.section(pattern.range(1));
    } else {
        if (yexPatExtraction_debugLevel > 1) {
            cerr << "Regex match failed to match string "
                 << extractedValue << " with pattern " << patStr <<
endl;
        }
        extractedValue = OTC_String::undefinedString();
    }
} else {
    cerr << "ERROR: invalid pattern for extraction: " << op_ <<
endl;
}

} else if (op_ != "") {
    // treat anything else like op_ == "", but warn if
    // bad value of op_
    cerr << "WARNING: Unrecognized operator seen for extraction, op
is: " << op_ << endl;
}

if (!extractedValue.isUndefined()) {
    extractedValue = extractedValue.trim();

    if (removeSpace_) {
        extractedValue = extractedValue.replace(" ", "");
    }

    if (yexPatExtraction_debugLevel > 1) {
        cerr << "adding tag extracted val " <<
            to_ << "=" << extractedValue << endl;
    }
    if (hash.contains(to_) && default_ == "") {
        const yutString &oldValue = hash.get(to_);
        yutString newValue = oldValue + " " + extractedValue;
        hash.set(to_, newValue);
    } else
        hash.set(to_, extractedValue);
}
}
}

```

```

patExtraction.h

/**
 * A pattern extraction is associated with a patElem and gives info
 * needed to extract data from a tag
 **
 */

#ifndef yexPatExtraction_h
#define yexPatExtraction_h

#include <iostream.h>
#include "yut/string.h"

class yhpHtmlToken;

class yexPatExtraction {
    friend ostream& operator<<(ostream& os, yexPatExtraction&
patExtraction) {
        os << "Extraction, from= " << patExtraction.from_ << ". to= " <<
patExtraction.to_ << ". op= " << patExtraction.op_;
        return os;
    }

private:
public:
    yutString from_;
    yutString to_;
    yutString op_;
    int opCode_;
    yutString default_; // default value. These are prefilled in the
hash
    // and overwritten if the extraction of that variable succeeds (so
    // that variable is never appended to if you use a default). Also,
    // the returned hash will have that value even if the the pattern.
did
    // not match

    bool removeSpace_; // means remove all whitespace from extracted
value.
    // Constructor
    //yexPatExtraction();
    void extract(yutHash& hash, yhpHtmlToken* token);
}; // class yexPatExtraction {

#endif // yexPatExtraction_h

```

patFilter.cc

```
/**
 * A pattern filter is associated with a patElem and acts as
 * a filter for determining if the input token matches the pattern
 * element.
 *
 * It supports the following operators:
 * equal: requires that the value must equal the filter operand
 * include: requires that the value must include the operand as a
 * substring
 * exclude: requires that the value must not include the operand as a
 * substring
 * include-or: requires that the value must include at least one of the
 * words in the operand
 * regex: requires the that value must satisfy the regular expression
 * contained in the operand
 * regex-neg: requires that the value must not satisfy the regular
 * expression contained in the operand
 */
**

#include <OTC/debug/assert.h>
#include "patFilter.h"
#include "htmlToken.h"
#include "htmlTagToken.h"
#include <OTC/Regexp.hh>

int yexPatFilter_debugLevel = 0;

ostream& operator<<(ostream& os, yexPatFilter& patFilter) {
    os << "Filter, op= " << patFilter.op_ << ". attribute= " <<
    patFilter.attribute_ << ". value= " << patFilter.value_ << ". Opcode is
    " << patFilter.opCode_ << ". Unknown opcode is " <<
    patFilter.UNKNOWN_OP_CODE;
    return os;
}

// constructor
yexPatFilter::yexPatFilter() {
    //cerr << "patFilter constr called" << endl;
} // constructor

void yexPatFilter::setOp(yutString opStr) {
    op_ = opStr;

    if (op_ == "include") {
        opCode_ = INCLUDE_OP_CODE;
    } else if (op_ == "equal") {
        opCode_ = EQUAL_OP_CODE;
    } else if (op_ == "exclude") {
        opCode_ = EXCLUDE_OP_CODE;
    } else if (op_ == "equal-or") {
        opCode_ = EQUAL_OR_OP_CODE;
    } //splitValue();
}
```

```

    } else if (op_ == "include-or") {
        opCode_ = INCLUDE_OR_OP_CODE;
        //splitValue();
    } else if (op_ == "regex") {
        opCode_ = REGEX_OP_CODE;
    } else if (op_ == "regex-neg") {
        opCode_ = REGEX_NEG_OP_CODE;
    } else {
        cerr << "Error: unknown op-code seen in: " << *this << endl;
        opCode_ = UNKNOWN_OP_CODE;
    }
}

// setOp()

// Returns true if the given token passes this filter
//
bool yexPatFilter::passesFilter(yhpHtmlToken* token) {

    if (yexPatFilter_debugLevel > 0) {
        cerr << "yexPatFilter::passesFilter(), filter is " << *this <<
endl;
        cerr << "token is " << token->getContent() << endl;
    }

    .yutString curValue = "";
    // no matter what kind of tag, return content for $this
    if (attribute_ == "$this") {
        curValue = token->getContent();
    } else if (token->type() == yhpHtmlToken::TAG_TYPE) {
        curValue = ((yhpHtmlTagToken*) token)-
>getAttributeValue(attribute_);
    } else if (token->type() == yhpHtmlToken::TEXT_TYPE) {
        if (attribute_ == "text") {
            curValue = token->getContent();
        } else {
            return false;
        }
    } else if (token->type() == yhpHtmlToken::COMMENT_TYPE) {
        if (attribute_ == "comment") {
            curValue = token->getContent();
        } else {
            return false;
        }
    } else {
        OTCLIB_ASSERT(false); // unknown token type
    }

    curValue = curValue.trim();
    if (curValue == "") return false;

    if (yexPatFilter_debugLevel > 0) {
        cerr << "curValue = " << curValue << endl;
        cerr << "value_ = " << value_ << endl;
    }
}

```

```

switch (opCode_) {
    case INCLUDE_OP_CODE:
        return curValue.index(value_) >= 0;
    case EQUAL_OP_CODE:
        return curValue == value_;
    case EXCLUDE_OP_CODE:
        return curValue.index(value_) < 0;
    case REGEX_OP_CODE:
    case REGEX_NEG_OP_CODE:
        {
            OTC_Regexp pattern(value_);
            if (!pattern.isValid()) {
                cerr << "Invalid regex filter pattern: " << value_ << endl;
                return false;
            }

            if (yexPatFilter_debugLevel > 3) {
                cerr << "pattern = " << value_ << endl;
                cerr << "checking against string = " << curValue << endl;
                cerr << "pattern.match(curValue) = " <<
                pattern.match(curValue) << endl;
            }
            bool matches = pattern.match(curValue);
            return (opCode_ == REGEX_NEG_OP_CODE) ? !matches : matches;
        }
    case UNKNOWN_OP_CODE:
    default:
        return false;
} // switch
return false; // should never get here
} // passesFilter()

```

patFilter.h

```
/**
 * A pattern filter is associated with a patElem and acts as
 * a filter for determining if the input token matches the pattern
 * element.
 *
 * It supports the following operators:
 * equal: requires that the value must equal the filter operand
 * include: requires that the value must include the operand as a
 substring
 * exclude: requires that the value must not include the operand as a
 substring
 * include-or: requires that the value must include at least one of the
 words in the operand
 * regex: requires the that value must satisfy the regular expression
 contained in the operand
 * regex-neg: requires that the value must not satisfy the regular
 expression contained in the operand
 **
 */

#ifndef yexPatFilter_h
#define yexPatFilter_h

#include <iostream.h>
#include "yut/string.h"

extern int yexPatFilter_debugLevel;

class yhpHtmlToken;

class yexPatFilter {
    friend ostream& operator<<(ostream& os, yexPatFilter& patFilter);

private:
    yutString op_;

public:
    const int UNKNOWN_OP_CODE = 0;
    const int INCLUDE_OP_CODE = 1;
    const int EQUAL_OP_CODE = 2;
    const int EXCLUDE_OP_CODE = 3;
    const int EQUAL_OR_OP_CODE = 4;
    const int INCLUDE_OR_OP_CODE = 5;
    const int REGEX_OP_CODE = 6;
    const int REGEX_NEG_OP_CODE = 7;

    yutString attribute_; // attribute name
    yutString value_;
    int opCode_;

    //Vector valueList;
    //Pattern pattern;
};
```


patMatcher.cc

```
/**
 * This class is the main entry point for the approximate string matching
 * (dynamic programming) algorithm. It matches a stream of htmlTokens
 * against a pattern, and extracts info out of the tokens.
 */

// stdlib is for atoi()
#include <stdlib.h>
#include <fstream.h>
#include <iostream.h>
#include <vector.h>
#include "htmlTokenizer.h"
#include "htmlTagToken.h"
#include "table.h"
#include "patMatcher.h"
#include "patFilter.h"
#include "patExtraction.h"
#include "patElem.h"
#include "tokFilter.h"
#include <OTC/debug/assert.h>

int debugLevel_ = 0; // set this before the matcher is
// constructed to get all debug (e.g. one that checks if
// stream had any tokens (strmToker.complainIfNoToks_)

// shared (by two methods) part of constructor, so put in own
// function. This is called in the constr, so is called
// before init(), which reads in config files
//

yexTablePool yexPatMatcher::tablePool;

void yexPatMatcher::reallyInit() {

    tableP_ = NULL;
    htmlTokerP_ = new yhpHtmlToker();
    inputEnded_ = false;
    configTableSize_ = 0;
    curLookahead_ = 0;
    curMatchCell_ = NULL;
    curMatchCandidate_ = NULL;
    lastMatchedIndex_ = -1;
    curCol_ = 0; // we really start at 1 since 0 is for epsilon (empty
string)
    // i.e. this is bumped up 1 before used
    inputStreamSet = false;
    patName_ = "";
    assertCol_ = 0;
    assertPat_ = "";
} // reallyInit

// Call this after determining there is a match on current col
```

```

// Extracted vals passes out in hash.
void yexPatMatcher::getMatchResult(yutHash& hash) {
    OTCLIB_ASSERT(curMatchCell_);

    // preset hash with default values.
    //
    setDefaultValues(hash);

    getMatchResultRecur(hash, curMatchCell_);
}

// set default values from <extraction> tags into hash
void yexPatMatcher::setDefaultValues(yutHash& hash) {

    for (unsigned int i = 1; i < (unsigned int)tableP_->getNumRows();
i++) {
        for (unsigned int j = 0; j < tableP_->patElems_[i]-
>extractions_.size(); j++) {
            yexPatExtraction* extraction = tableP_->patElems_[i]-
>extractions_[j];
            if (extraction->default_ != "") {
                hash.set(extraction->to_, extraction->default_);
            }
        }
    }
} // setDefaultValues

// only call this from getMatchResult(yutHash), since this
// is recursive call, and that one has code that needs to
// be called exactly once for an item
void yexPatMatcher::getMatchResultRecur(yutHash& hash, yexCell* cell) {

    //yexCell* cell = curMatchCell_;
    //OTCLIB_ASSERT(cell);

    if (!cell || cell->row == 0) return; // ends recursion when get to
null

    // recursively call on predecessor
    //
    getMatchResultRecur(hash, tableP_->predCellInDirection(cell,
(yexTable_directionT) cell->predDirec));

    // now deal with current cell
    if (debugLevel_ > 1) tableP_->dumpCell(cell);

    if (cell->predDirec == yexTable_DIAG) {

        yhpHtmlToken* token = tableP_->getToken(cell->col);
        if (token == NULL) return;

        yexPatElem* patElem = tableP_->getPatElem(cell->row);
        if (patElem == NULL) return;
    }
}

```

```

        patElem->extract(hash, token);
    }

} // getMatchResultRecur

// constructor
yexPatMatcher::yexPatMatcher(istream& istream) {
    reallyInit();
    setInputStream(istream);
};

yexPatMatcher::yexPatMatcher() {
    reallyInit();
}

// destr
yexPatMatcher::~yexPatMatcher() {
    if (htmlTokenizerP_) delete htmlTokenizerP_;
    //if (tableP_) delete tableP_;
    tableP_ = NULL;

    for (int i = 0; i < (int) patElems->size(); i++) {
        delete ((*patElems)[i]);
        (*patElems)[i] = NULL;
    }
    delete patElems;
}

void yexPatMatcher::setInputStream(istream& istream) {

    if (inputStreamSet) { // if this is 2nd or more time you've reset
        // input stream (ie you are resetting it).
        if (htmlTokenizerP_) delete htmlTokenizerP_;
        htmlTokenizerP_ = new yhpHtmlTokenizer();
        htmlTokenizerP_->outputWhitespaceTextToks(false);
        htmlTokenizerP_->setFixMissingCloseQuote(true);    reset();
    }
    htmlTokenizerP_->setInputStream(istream);
    inputStreamSet = true;
};

// Call this if you change the input stream to reset the patternMatcher
// See reset rules above.

void yexPatMatcher::reset() {

    if (tableP_) tableP_->reset();
    curLookahead_ = 0;
    curMatchCell_ = NULL;
    curMatchCandidate_ = NULL;
    inputEnded_ = false;
    lastMatchedIndex_ = -1;
    curCol_ = 0;
}

```

```

bool yexPatMatcher::inGroupMatch() {
    return patName_ != ""; // currently only named if in group match
}

// Read in config pattern and filter file, build table.
// if usePatdescr, then read patdescr file instead of config, pattern,
// and patternStream is padescr file. Returns false iff error
// (e.g. missing config tag in patdescr file).
//
bool yexPatMatcher::init(istream& patternStream, istream& configStream,
                        istream& filterStream, bool usePatdescr = false) {
    /***LEV
    //int pcols;

    htmlTokenizerP->outputWhitespaceTextToks(false); // wasfalse
    htmlTokenizerP->setFixMissingCloseQuote(true);

    htmlTokenizerP->setComplainIfNoToks(true);

    if (debugLevel_ > 0) {
        otagfile_.open("tag.out", ios::out);
        if (!otagfile_) {
            cerr << "ERROR: Could not open tag.out for output." << endl;
        } else {
            cerr << "Outputting tags to tag.out...\n";
        }
    }

    if (filterStream) {
        readInFilter(filterStream);
    } else if (debugLevel_ > 0) {
        cerr << "Filter file not read since not present" << endl;
    }

    /***LEV
    //yexPatElem_Vect* patElems;

    if (usePatdescr) {
        patElems = readInPatdescr(patternStream);
        if (!patElems) {
            return false;
        }
    } else {
        patElems = readInConfig(patternStream, configStream);
    }

    if (configTableSize_ == 0) { // not set in config file
        pcols = patElems->size() + threshold_ + lookahead_ + 1;
        // cerr << "computed TableSize is " << pcols << endl;
    } else {
        pcols = configTableSize_;
        OTCLIB_ASSERT(pcols >= (int) (patElems->size() + threshold_ +
        lookahead_ + 1));
    }
}

```

```

numRows_ = patElems->size()+1;
/**LEV
//tableP_ = new yexTable(numRows_, pcols, patElems);

//cerr << "DIMENSIONS = " << numRows_ << " " << pcols << endl;

#ifdef TABLE_STAT

//cerr << "TABLE CREATED" << endl;
//cerr << "TABLE SIZE = " << sizeof(*tableP_) << endl;

//for(int i = 0; i < (int)patElems->size(); i++) {
//    cerr << "PATTERN TOKEN CREATED" << endl;
//    cerr << "PATTERN TOKEN SIZE = " << sizeof((*patElems)[i]) <<
endl;
//}

#endif

//delete patElems;
return true;
} // init

// This version takes a dir and figures out if should use patdescr file
// Returns true if OK, false if error.
bool yexPatMatcher::init(yutString patternDir) {

    yutString filterPath = patternDir + "/filter";
    ifstream filterFile(filterPath, ios::in);
    /*
    if (!filterFile) {
        cerr << "ERROR: could not open filter " << filterPath << endl;
        return false;
    }
    */
    yutString patdescrPath = patternDir + "/patdescr";
    ifstream patdescrFile(patdescrPath, ios::in);
    if (patdescrFile) {
        ifstream configFile; // dummy
        return init(patdescrFile, configFile, filterFile, true
/*usePatdescr*/);
    } else {
        yutString patternPath = patternDir + "/pattern";
        ifstream patternFile(patternPath, ios::in);
        if (!patternFile) {
            cerr << "ERROR: could not open pattern " << patternPath <<
endl;
            return false;
        }
        yutString configPath = patternDir + "/config";
        ifstream configFile(configPath, ios::in);
        if (!configFile) {
            cerr << "ERROR: could not open config " << configPath << endl;
            return false;
        }
        return init(patternFile, configFile, filterFile);
    }
}

```

```

    }
} // init (patternDir)

// process token, return true if there is a match of the pattern
// (also increments curCol_)
bool yexPatMatcher::processToken(yhpHtmlToken* tokenP) {

    if (otagfile_ && tokenP) {
        if (tokFilter_.filterOut(tokenP)) {
            //cerr << "*** " << tokenP->toWellFormedString() << endl;
            otagfile_ << "*** " << tokenP->toWellFormedString() << endl;
        } else {
            //cerr << curCol_ << " " << tokenP->getContent() << endl;
            otagfile_ << curCol_ << " " << tokenP->getContent() << endl;
        }
    }

    if (tableP_ == NULL) return false;

    if (tokenP == NULL) {
        return matchFound(true /*inputEnded*/);
    }

    if (tokFilter_.filterOut(tokenP)) {
        tokenP->decrRefCount(); // will delete if last reference
        return false;
    }

    curCol_++;
    //cerr << "curcol, logCols " << curCol_ << " " << tableP_-
>getNumLogCols() << endl;
    if (curCol_ >= tableP_->getNumLogCols()) {
        //cerr << "shifting table..." << endl;
        tableP_->shiftTable(curCol_);
    }

    tableP_->setToken(curCol_, tokenP);
    for (int row = 1; row < numRows_; row++) {
        tableP_->computeCellCost(row, curCol_);
    }

#ifdef _HASH_ENABLED_

    if (tokenP->type() == yhpHtmlToken::TAG_TYPE &&
        ((yhpHtmlTagToken*)tokenP)->getName() == "assert") {

#else

    if (tokenP->type() == yhpHtmlToken::TAG_TYPE &&
        ((yhpHtmlTagToken*)tokenP)->getType() ==
        yhpHtmlTagToken::hash("assert")) {

#endif

        assertCol_ = curCol_ + 1;
    }
}

```

```

        assertPat_ = ((yhpHtmlTagToken*)tokenP)->getAttributeValue("pat");
    }

    bool matched = matchFound(false /*inputEnded*/);
    if (matched && debugLevel_ > 0) {
        cerr << "After match (patMatcher.cc proceToken), here is table:"
<< endl;
        tableP_->dumpTable();
    }

    return matched;
} // processToken

// If currently on an area of html that has been "asserted" to
// to match, assert if it didn't
void yexPatMatcher::checkAssertTag() {
    if (!assertCol_) return; // not in region

    // if in group mode, need to have pat name match for assert
    if (assertPat_ != "" && patName_ != "" && assertPat_ != patName_) {
        return;
    }
    if (curCol_ > assertCol_ + numRows_ + threshold_ + lookahead_ + 1) {
        // complain, dump table
        cerr << "*** Match Assert Failed. " << endl;
        cerr << "assertCol_ = " << assertCol_ << endl;
        cerr << "curCol_ = " << curCol_ << endl;
        cerr << "assertPat_ = " << assertPat_ << endl;
        tableP_->dumpTable();
        assertCol_ = 0;
    }
} // checkAssertTag

// Returns true if there is a match at curCol_
//
bool yexPatMatcher::matchFound(bool inputEnded) {

    yexCell* cell = tableP_->getCell(numRows_-1, curCol_);
    OTCLIB_ASSERT(cell->col == curCol_);
    OTCLIB_ASSERT(cell && cell->getCost() >= 0); // check that cost
    been calc'ed

    // If currently on an area of html that has been "asserted" to
    // to match, assert if it didn't (pattern debug tool)
    checkAssertTag();

    if (curMatchCandidate_ == NULL) {
        if (cell->index > lastMatchedIndex_ && cell->getCost() <=
threshold_) {

            curMatchCandidate_ = cell;
        } else {
            // if there is no candidate and this cell can't be one,
            // then no match, return
            return false;
        }
    }
}

```

```

    if (debugLevel_ > 2) {
        cerr << "matchfound, cur col: " << curCol_ << " matchCell col " <<
            curMatchCandidate_>col << ". Lookahead=" << curLookahead_<<
        " of " << lookahead_<< endl;
        cerr << "curCell cost=" << cell->getCost() << ". matchCell cost="
        "
            << curMatchCandidate_>getCost() << ". curMatch index=" <<
            << curMatchCandidate_>index << endl << endl;
    }

    if (cell->getCost() <= threshold_ && cell->getCost() <=
        curMatchCandidate_>getCost() &&
        cell->index > lastMatchedIndex_) {
        // favor longer matches for tie break
        curMatchCandidate_ = cell;
        curLookahead_ = 0;
    } else {
        curLookahead_++;
    }
    /*
    if (curMatchCandidate_>getCost() <= threshold_) {
        cerr << "okie under thresh, " << " curCol_ " << curCol_ <<
        " matchCand col " << curMatchCandidate_>col << " lookie: " <<
        (curLookahead_ >= lookahead_ || inputEnded) <<
        " index check is " << (curMatchCandidate_>index >
        lastMatchedIndex_) << " curindex " <<
        curMatchCandidate_>index << " lastInd " <<
        lastMatchedIndex_ << endl;
        cerr << " " << endl;
    }
    */

    if ((curMatchCandidate_>getCost() <= threshold_) &&
        (curLookahead_ >= lookahead_ || inputEnded) &&
        (curMatchCandidate_>index > lastMatchedIndex_)) {

        curMatchCell_ = curMatchCandidate_;
        lastMatchCell_ = curMatchCell_;
        lastMatchedIndex_ = curMatchCandidate_>index;
        curMatchCandidate_ = NULL;
        curLookahead_ = 0;
        if (debugLevel_ > 0) cerr << endl << "**** Found match! curCol is "
        << curCol_ << ". MatchCol is " << curMatchCell_>col << endl << endl;

        // if (assertPat_ == "" || patName_ == "" || assertPat_ ==
        patName_) {

            if (assertCol_ > 0 /* && debugLevel_ >= 1 */ &&
                (assertPat_ == "" || patName_ == "" || assertPat_ ==
                patName_)) {
                cerr << "**** Match Assert Succeeded. " << endl;
                cerr << "assertCol_ = " << assertCol_ << endl;
                cerr << "curCol_ = " << curCol_ << endl;
                cerr << "assertPat_ = " << assertPat_ << endl;
            }
        }
    }

```

```

        assertCol_ = 0; // reset since saw pattern
        assertPat_ = "";
        return true;
    } else
        return false;
} // matchFound

// returns true and puts next item (set of name-value pairs) into hash
// if match, else returns false indicating at EOF and no more matches
//
bool yexPatMatcher::nextItem(yutHash& resultHash) {

    if (tableP_ == NULL) return false;

    yhpHtmlToken* tokenP;

    while (true) {
        if (inputEnded_) {
            return false;
        }
        tokenP = htmlTokerP_>nextToken();

        if (processToken(tokenP)) {
            getMatchResult(resultHash);
            return true;
        }
        if (debugLevel_ > 2 && curCol_ % tableP_>getNumPhysCols() ==
            tableP_>getNumPhysCols()-1) {
            tableP_>dumpTable();
        }

        if (tokenP == NULL) {
            inputEnded_ = true;
            if (debugLevel_ > 2) tableP_>dumpTable();
            return false;
        }
    }
} // nextItem()

void yexPatMatcher::readInFilter(istream& in) {
    const BUFSIZE = 100;
    char buf[BUFSIZE];
    yutString line;

    while (in.getline(buf, BUFSIZE)) {
        line = buf;
        line = line.trim(); // Remove trailing and leading white space
        //cerr << "buf" << buf << endl;
        if (line.length() == 0) continue; // skip over an empty line
        if (line[0] == '+') {

```

```

        tokFilter_.addFilterToken(line.from(1));
        continue;
    }
    if (line[0] == '-') {
        if (line == "-comment") {
            //cerr << "keeping comment due to -comment" << endl;
            tokFilter_.keepCommentToken = true;
        } else {
            tokFilter_.removeFilterToken(line.from(1));
            //cerr << "removing = " << line.from(1) << endl;
        }
        continue;
    }
} // while
} // readInFilter

//
yexPatElem_Vect* yexPatMatcher::readInConfig(istream& patStream,
istream& configStream) {
    yhpHtmlToken* tokenP;
    yexPatElem_Vect* patElems = new yexPatElem_Vect; // to be returned

    // Read in pattern file
    yhpHtmlToker patToker(patStream);
    patToker.outputWhitespaceTextToks(false);

    while ((tokenP = patToker.nextToken())) {
        if (tokFilter_.filterOut(tokenP)) {
            delete tokenP; // config file token
        } else {
            yexPatElem* patElem = new yexPatElem(tokenP);
            patElems->push_back(patElem); // add to list
        }
    } // while

    if (debugLevel_ > 1) cerr << "after reading pattern, patElems-
size() = " << patElems->size() << endl;

    // Read the config file and annotate the patElems that are
    referenced
    // with <state> tags
    //
    yhpHtmlToker configToker(configStream);
    yexPatElem* currentPE = NULL;

    while ((tokenP = configToker.nextToken())) {
        if (tokenP->type() != yhpHtmlToken::TAG_TYPE) {
            delete tokenP; // config file token
            continue;
        }

        // actually copy the tag, and delete the pointer to the
        // dynamic one, to gaurantee no mem leaks
        yhpHtmlTagToken* tagP = (yhpHtmlTagToken*) tokenP;
        yhpHtmlTagToken tag;

```

```

tag = *tagP;
delete tokenP; // config file token

#ifdef _HASH_ENABLED_

yutString name = tag.getName();
if (name == "state") {

#else

int tagType = tag.getType();
if (tagType == yhpHtmlTagToken::hash("state")) {

#endif

yutString posStr = tag.getAttributeValue("position");
int pos = atoi(posStr);
currentPE = (*patElems)[pos];

#ifdef _HASH_ENABLED_

} else if (name == "config") {

#else

} else if (tagType == yhpHtmlTagToken::hash("config")) {

#endif

threshold_ = atoi(tag.getAttributeValue("threshold"));
lookahead_ = atoi(tag.getAttributeValue("lookahead"));
// can't look ahead in group match because if another
// pat sequentially after it, it will consume too
// many tokens, ie the ones the next guy should have
if (inGroupMatch()) {
    lookahead_ = 0;
}

configTableSize_ = atoi(tag.getAttributeValue("tablesize"));

if (debugLevel_ > 1) {
    cerr << "threshold_ = " << threshold_ << endl;
    cerr << "lookahead_ = " << lookahead_ << endl;
    cerr << "configTableSize_ = " << configTableSize_ << endl;
}

#ifdef _HASH_ENABLED_

} else if ((name == "extraction" || name == "filter") && currentPE
== NULL) {

#else

} else if ((tagType == yhpHtmlTagToken::hash("extraction") ||
tagType == yhpHtmlTagToken::hash("filter")) &&
currentPE == NULL) {

```

```

#endif

        errorMsg("Internal error, missing patternElem. Tag:" +
            tag.getContent());
        return NULL;
    } else {
        // apply this config tag to the current PE.
        currentPE->applyConfigTag(tag);
    }

} // while

return patElems;

} // readInConfig

bool isConfigTag(yhpHtmlToken* tokenP) {
    if (tokenP->type() != yhpHtmlToken::TAG_TYPE) {
        return false;
    }
}

#ifdef _HASH_ENABLED_

    yutString name = ((yhpHtmlTagToken*) tokenP)->getName();
    return (name == "config" || name == "score" ||
        name == "filter" || name == "extraction" ||
        name == "state" || name == "merchant");

#else

    int tagType = ((yhpHtmlTagToken*) tokenP)->getType();
    return (tagType == yhpHtmlTagToken::hash("config") ||
        tagType == yhpHtmlTagToken::hash("score") ||
        tagType == yhpHtmlTagToken::hash("filter") ||
        tagType == yhpHtmlTagToken::hash("extraction") ||
        tagType == yhpHtmlTagToken::hash("state") ||
        tagType == yhpHtmlTagToken::hash("merchant"));

#endif

} // isConfigTag

//
yexPatElem_Vect* yexPatMatcher::readInPatdescr(istream& patdescrStream)
{
    //ifstream& infile("test.dat", ios::in);
    bool sawConfigTag = false; // we fail if not seen
    yhpHtmlToken* tokenP;
    yexPatElem_Vect* patElems = new yexPatElem_Vect; // to be returned

    yhpHtmlToken patToken(patdescrStream);
    patToken.outputWhitespaceTextToks(false);

    // Read the config file and annotate the patElems that are
    referenced

```

```

// with <state> tags
//
yexPatElem* currentPE = NULL;

// We want to check for the error condition where there
// is a tag that is filtered (e.g. <font>) followed by
// a config tag like extract.
yutString lastTagFiltered = "";

while ((tokenP = patToker.nextToken())) {
    // if it's not a config tag, treat as part of the pattern
    if (!isConfigTag(tokenP)) {
        if (!tokFilter_.filterOut(tokenP)) {
            lastTagFiltered = "";
            currentPE = new yexPatElem(tokenP);
            // xxx
            patElems->push_back(currentPE); // add to list
        } else {
            lastTagFiltered = tokenP->getContent(); // this tag was
filtered
            delete tokenP; // config tag
        }
    } else { // it's a config tag
        if (lastTagFiltered != "") {
            errorMsg("Error in patdescr file. Had a control tag after a
filtered tag.\nFiltered tag is:\n" + lastTagFiltered + "\nControl tag
is:\n" + tokenP->getContent());
            return NULL;
        }
        // this was part of isConfigTag() check
        OTCLIB_ASSERT(tokenP->type() == yhpHtmlToken::TAG_TYPE);
        yhpHtmlTagToken* tagP = (yhpHtmlTagToken*) tokenP;

#ifdef _HASH_ENABLED_

        yutString name = tagP->getName();

        if (name == "state") {
            warnMsg("Saw state tag in tagdescr -- not allowed. Tag:" +
                tagP->getContent());
            return NULL;
        } else if (name == "config") {

#else

        int tagType = tagP->getType();

        if (tagType == yhpHtmlTagToken::hash("state")) {
            warnMsg("Saw state tag in tagdescr -- not allowed. Tag:" +
                tagP->getContent());
            return NULL;
        } else if (tagType == yhpHtmlTagToken::hash("config")) {

#endif

        sawConfigTag = true;
    }
}

```

```

threshold_ = atoi(tagP->getAttributeValue("threshold"));
lookahead_ = atoi(tagP->getAttributeValue("lookahead"));
// can't lookahead in group match because if another
// pat sequentially after it, it will consume too
// many tokens, ie the ones the next guy should have
if (inGroupMatch()) {
    lookahead_ = 0;
}

configTableSize_ = atoi(tagP->getAttributeValue("tablesize"));

if (debugLevel_ > 1) {
    cerr << "threshold_ = " << threshold_ << endl;
    cerr << "lookahead_ = " << lookahead_ << endl;
    cerr << "configTableSize_ = " << configTableSize_ << endl;
}
// put on list of config tags to output if write out patdesc
configTags_.push_back(tagP);

#ifdef _HASH_ENABLED_
} else if (name == "merchant") {

#else
} else if (tagType == yhpHtmlTagToken::hash("merchant")) {

#endif

// put on list of config tags to output if write out patdesc
configTags_.push_back(tagP);

#ifdef _HASH_ENABLED_
} else if ((name == "extraction" || name == "filter") && currentPE
== NULL) {

#else
} else if ((tagType == yhpHtmlTagToken::hash("extraction") ||
tagType == yhpHtmlTagToken::hash("filter")) && currentPE
== NULL) {

#endif

errorMsg("Internal error, missing patternElem. Tag: " +
tagP->getContent());
return NULL;
} else {
// first save the tag in case we write out
currentPE->configTags_.push_back(tagP);

// apply this config tag to the current PE.
currentPE->applyConfigTag(*tagP);
}
delete tokenP; // config tag
} // it's a config tag

```

```

    } // while

    if (!sawConfigTag) {
        errorMsg("Required <config> tag missing in patdescr file.");
        return NULL;
    }
    return patElems;
} // readInPatdescr

// write out pattern and config files
bool yexPatMatcher::writeOutConfig(yutString patternPath, yutString
configPath) {
    ofstream patternFile(patternPath, ios::out);
    if (!patternFile) {
        cerr << "ERROR: could not open file for output: " << patternPath <<
endl;
        return false;
    }
    ofstream configFile(configPath, ios::out);
    if (!configFile) {
        cerr << "ERROR: could not open file for output: " << configPath <<
endl;
        return false;
    }

    // output config tags that are at begining of config file
    for (unsigned int i = 0; i < configTags_.size(); i++) {
        configFile << configTags_[i]->toWellFormedString() << endl;
    }

    // output tags associated with a given patElem
    for (unsigned int i = 1; i < (unsigned int)tableP_->getNumRows(); i++)
    {
        yhpHtmlToken* tokenP = tableP_->patElems_[i]->tokenP_;
        patternFile << tokenP->toWellFormedString() << endl;;

        bool gaveState = false;
        for (unsigned int j = 0; j < tableP_->patElems_[i]-
>configTags_.size(); j++) {
            if (!gaveState) {
                configFile << "\n<state position=\"" << i-1 << ">" << endl;
                gaveState = true;
            }
            configFile << tableP_->patElems_[i]->configTags_[j]-
>toWellFormedString() << endl;
        }
        return true;
    } // writeOutConfig

void yexPatMatcher::warnMsg(yutString msg) {
    yhpHtmlToker::prefixMsg(cerr, "warn:yex: ", msg);
}

void yexPatMatcher::errorMsg(yutString msg) {

```


patMatcher.h

```
/**
 * This class is the main entry point for the approximate string matching
 * (dynamic programming) algorithm. It matches a stream of htmlTokens
 * against a pattern, and extracts info out of the tokens.
 */

#ifndef yexPatMatcher_h
#define yexPatMatcher_h

#include "iostream.h"
#include "fstream.h"
#include "yhp/htmlTokenizer.h"
#include "yex/table.h"
#include "yex/tokFilter.h"
#include "yex/tablePool.h"

extern int debugLevel_;

class yexPatMatcher {
private:
    int numRows_;           // num rows in table
    int curCol_;            // current col in table
    yexTokFilter tokFilter_;
    int threshold_;
    int lookahead_;         // max lookahead dist (from config file)
    int curLookahead_;      // current lookahead dist
    yexCell* curMatchCandidate_; // this one matches, but we look ahead
    by lookahead cols
    yexCell* curMatchCell_;  // real returned match
    yexCell* lastMatchCell_; // This is same as curMatchCell, but
    not reset when          // reset() is called. So we can
    reset patMatchers        // right after a match when doing
    parallel matching,        // but this variable keeps state for
    outputting match
    int lastMatchedIndex_;
    int configTableSize_;   // table size from config file (can use for
    debug)
    yhpHtmlTokenizer* htmlTokenizerP_;
    bool inputEnded_;
    bool inputStreamSet;    // true if has ever been set
    ofstream otagfile_;     // debug dump of input file tags ("tag.out",
    ios::out);

    int assertCol_;
    yutString assertPat_;

    // holds config tags <merchant> and <config> for output later
    vector<yhpHtmlTagToken*> configTags_;
};
```

```

    yexPatElem_Vect* readInConfig(istream& patStream, istream&
configStream);
    yexPatElem_Vect* readInPatdescr(istream& patdescrStream);
    void readInFilter(istream& in);
    void reallyInit();
    void yexPatMatcher::getMatchResultRecur(yutHash& hash, yexCell*
cell);
    void yexPatMatcher::checkAssertTag();

    /***LEV
    int pcols;
    yexPatElem_Vect* patElems;

    static yexTablePool tablePool;

public:
    yexTable* tableP_;
    yutString patName_; // set if in group mode, else "". mainly for
debug

    bool writeOutConfig(yutString patternPath, yutString configPath);
    void reset();
    void setDebugLevel(int level) {debugLevel_ = level;};
    yexPatMatcher(istream& instream);
    yexPatMatcher();
    ~yexPatMatcher();
    void setInputStream(istream& istream);
    static void test();
    bool yexPatMatcher::init(istream& patternStream, istream&
configStream,
                           istream& filterStream, bool usePatdescr = false);
    bool yexPatMatcher::init(yutString patternDir);
    bool processToken(yhpHtmlToken* tokenP);
    bool yexPatMatcher::matchFound(bool inputEnded);
    // Call this one from external:
    void yexPatMatcher::getMatchResult(yutHash& hash);
    bool yexPatMatcher::nextItem(yutHash& resultHash);
    void yexPatMatcher::setDefaultValues(yutHash& hash);
    bool yexPatMatcher::inGroupMatch();
    static void warnMsg(yutString msg);
    static void errorMsg(yutString msg);

    void yexPatMatcher::startProcessing(bool start);

}; // class yexPatMatcher

#endif // yexPatMatcher_h

```

```

patNode.cc

/* $Header: */

/**
 * Stores a node in the pattern group graph.  Graph is a DAG. *
 **/

#include <vector.h>
#include "patNode.h"

// Constr
yexPatNode::yexPatNode() {

    leafPatMatcher_ = NULL;
    isMultiMatch_ = false; // only set to true if is patMatcher node and
    is multi
    parent_ = NULL;
}

// Follows graph down to the exit node.
yexPatNode* yexPatNode::exitNode() {
    // Just follow first children down at each level until no children.
    //
    yexPatNode* pnode = this;
    while (true) {
        if (pnode->children_.size() < 1) {
            return pnode;
        }
        pnode = pnode->children_[0]; // follow 1st child
    }
} // exitNode

void yexPatNode::addChild(yexPatNode* child) {

    children_.push_back(child);
    // set the parent of the child to be this. OK if it is already
    // set to another parent.
    child->parent_ = this;
} // addChild

```

patNode.h

```
/* $Header: */

#ifndef yexPatNode_h
#define yexPatNode_h

#include "yut/string.h"

class yexPatMatcher;

class yexPatNode {

private:
    yexPatMatcher* leafPatMatcher_; // pattern of this node if it is a
    "leaf" else null.
    yexPatNode* parent_; // points to one (arbitrary which) of parents

public:
    yutString name_; // for debug. Name of pat or patGroup
    vector<yexPatNode*> children_;
    bool isMultiMatch_; // true if this pattern can match many times
    bool isIntertwine_;

    yexPatNode* exitNode();
    void addChild(yexPatNode* child);
    yexPatMatcher* getLeafPatMatcher(){return leafPatMatcher_;};
    yexPatNode* getParent(){return parent_;};
    void setLeafPatMatcher(yexPatMatcher* leafPatMatcher){leafPatMatcher_
= leafPatMatcher;};
    yexPatNode():yexPatNode();

}; // class yexPatNode

#endif // yexPatNode_h
```

table.cc

```
/**
 * This class represents the approximate string matching
 * table, used in the (dynamic programming) matching algorithm.
 */

#include <OTC/debug/assert.h>
#include "htmlTagToken.h"
#include "cell.h"
#include "patElem.h"
#include "patFilter.h"
#include "patExtraction.h"
#include "table.h"
#include "patMatcher.h"

// Constructor
//
yexTable::yexTable(int numRows, int numPhysCols, yexPatElem_Vect*
patElems) :
    cells(numRows * numPhysCols),
    tokens__(numPhysCols), patElems_(numRows) {

    maxRows = numRows;
    maxCols = numPhysCols;
    available = true;

    this->numRows = numRows;
    this->numPhysCols = numPhysCols;
    numLogCols = numPhysCols; // initial value

    OTCLIB_ASSERT(numRows >= (int) patElems->size()+1);

    for (int col = 0; col < numPhysCols; col++) {
        tokens__[col] = NULL;
    }

    // Keep row 0 special, -> epsilon
    patElems_[0] = new yexPatElem( new yhpHtmlTagToken((yutString)
" <epsilon>"));

    // Copy the patElems vector
    for (int i = 1; i < (int) patElems->size()+1; i++) {
        patElems_[i] = (*patElems)[i-1];
    }

    for (int row = 0; row < numRows; row++) {
        for (int col = 0; col < numPhysCols; col++) {
            yexCell* yexCellP = new yexCell(row, col);
            setCell(row, col, yexCellP);
            if (row == 0) {
                yexCellP->setCost(0);
                yexCellP->index = col;
            } else if (col == 0) {
                yexCellP->setCost(row);
            } else {

```

```

        yexCellP->setCost(-1); // means not set
    }
}

} // constructor

yexTable::yexTable(int numRows, int numPhysCols) :
    cells(numRows * numPhysCols),
    tokens__(numPhysCols), patElems_(numRows) {

    maxRows = numRows;
    maxCols = numPhysCols;
    available = true;

    this->numRows = numRows;
    this->numPhysCols = numPhysCols;
    numLogCols = numPhysCols; // initial value

    if (numRows > 0) patElems_[0] = NULL;

    for (int col = 0; col < numPhysCols; col++) {
        tokens__[col] = NULL;
    }

    for (int row = 0; row < numRows; row++) {
        for (int col = 0; col < numPhysCols; col++) {
            yexCell* yexCellP = new yexCell(row, col);
            setCell(row, col, yexCellP);
            if (row == 0) {
                yexCellP->setCost(0);
                yexCellP->index = col;
            } else if (col == 0) {
                yexCellP->setCost(row);
            } else {
                yexCellP->setCost(-1); // means not set
            }
        }
    }
}

// destructor
yexTable::~yexTable() {

    // delete cells
    for (int i = 0; i < maxRows * maxCols; i++) {
        if (cells[i]) delete cells[i];
        cells[i] = NULL;
    }

    // delete stored input tokens
    for (int col = 0; col < numPhysCols; col++) {
        if (tokens__[col]) tokens__[col]->decrRefCount(); // will delete
    }
    if last reference
        tokens__[col] = NULL;
}

```

```

    // delete the patElems vector
    for (int i = 0; i < (int) patElems_.size(); i++) {
        if (i == 0) delete patElems_[0];
        patElems_[i] = NULL;
    }
} // destructor

// Reset the table, clearing cells
void yexTable::reset() {
    numLogCols = numPhysCols;
    for (int row = 0; row < numRows; row++) {
        for (int col = 0; col < numPhysCols; col++) {
            yexCell* cell = getCell(row, col, false /*checkCol */);
            cell->reset(row, col);
            if (row == 0) {
                cell->setCost(0);
                cell->index = col;
            } else if (col == 0) {
                cell->setCost(row);
            }
        }
    }
} // reset()

// Shift table by one (increase) to include the passed
// column number.
//
void yexTable::shiftTable(int col) {
    // since before, numLogCols-1 was max col
    OTCLIB_ASSERT(col == numLogCols);

    // reallocate cells in this col
    for (int row = 0; row < numRows; row++) {
        // turn off checking because we are querying a cell
        // that has an old col value in it (so we can delete it)
        yexCell* cell = getCell(row, col, false /*checkCol */);
        // allocate new cell here
        if (cell) delete cell;
        cell = new yexCell(row, col);
        if (row == 0) {
            cell->setCost(0);
            cell->index = col;
        }
        setCell(row, col, cell);
    }
    numLogCols = col+1; // shift table to the right
} // shiftTable

// set cell at row 'row' and logical column 'col'
// will set the row can col in the cell
void yexTable::setCell(int row, int col, yexCell* cell) {
    int index = numPhysCols*row + col % numPhysCols;
    cells[index] = cell;
}

```

```

    cell->row = row;
    cell->col = col; // set logical col
}

// Return cell at row 'row' and logical column 'col'
//
yexCell* yexTable::getCell(int row, int col, bool checkCol = true) {
    OTCLIB_ASSERT( row < numRows);

    // The model is that the table grows by moving "right" to larger
    cols
    // over time. It is illegal to access a logical column once the
    // physical table has moved past it (it should never be necessary
    to
    // to so). This idea is encapsulated in the following check:

    if (checkCol && col < numLogCols-numPhysCols) {
        cerr << "Cannot access col " << col << ". Legal values are " <<
        numLogCols-numPhysCols << " to " << numLogCols-1 << endl;
        OTCLIB_ASSERT(false);
    }

    int index = numPhysCols*row + col % numPhysCols;
    yexCell* cell = cells[index];
    // Check that this is the expected col
    if (checkCol) {
        OTCLIB_ASSERT(col == cell->col);
    }
    return cell;
};

int yexTable::getCost(int row, int col) {
    yexCell* cell = getCell(row, col);
    if (col != cell->col) {
        cerr << "col = " << col << endl;
        cerr << "cell->col = " << cell->col << endl;
    }
    OTCLIB_ASSERT(col == cell->col); // check that at last write it was
    same logCol
    return cell->cost;
}

void yexTable::setCost(int row, int col, int cost) {
    yexCell* cell = getCell(row, col);

    cell->col = col; // write log col
    cell->cost = cost;
}

// Write logical col for this row, col, indicating a write
// is about to happen at this cell (error checking
// is done on col on read/write)
//
void yexTable::markColForWrite(int row, int col) {
    yexCell* cell = getCell(row, col);

```

```

        cell->col = col;                                // write log col
    }

// Return the preceding cell in the given direction fro the given cell.
// E.g. if direc = yexTable_HORIZ, then returns the cell to the left of
// it.
//
yexCell* yexTable::predCellInDirection(yexCell* cell,
yexTable_directionT direc) {
    if (direc == yexTable_HORIZ) {                    // get cost from one to left
        return getCell(cell->row, cell->col-1);
    } else if (direc == yexTable_VERT) {
        return getCell(cell->row-1, cell->col);
    } else if (direc == yexTable_DIAG) {
        return getCell(cell->row-1, cell->col-1);
    }
    return NULL;
} // predCellInDirection

// Calculate the cost to the given cell from the given direction.
// E.g. if direc = yexTable_HORIZ, then it is cost to this cell
// from the one on the left. Includes the predecessor cell cost
// plus the incremental directionCostTo cost to get to this cell
//
int yexTable::directionCostTo(yexCell* cell, yexTable_directionT direc)
{
    //cerr << "directionCostTo direc is " << direc << endl;
    //cerr << "r,c is " << cell->row << " " << cell->col << endl;

    OTCLIB_ASSERT((cell->row >= 1) && (cell->row < numRows));
    OTCLIB_ASSERT(cell->col >= 1);
    OTCLIB_ASSERT(patElems_[cell->row]);

    int predCellCost = predCellInDirection(cell, direc)->getCost();
    OTCLIB_ASSERT(predCellCost >= 0);

    if (direc == yexTable_HORIZ) {                    // get cost from one to left
        return patElems_[cell->row]->directionCost[direc] + predCellCost;
    } else if (direc == yexTable_VERT) { // get cost from one above
        return patElems_[cell->row] -> directionCost[direc] + predCellCost;
    } else if (direc == yexTable_DIAG) { // get cost from one left,above

        yhpHtmlToken* token = getToken(cell->col);
        OTCLIB_ASSERT(token);
        if (patElems_[cell->row]->matchesToken(token)) {
            return predCellCost; // perfect match -- zero additional cost
        } else {
            // usually quasi-infinite
            return patElems_[cell->row] -> directionCost[direc] +
predCellCost;
        }
    } else {
        OTCLIB_ASSERT(false); // bad direction passed
    }
}

```

```

    }

    return -1; // should never get here
} // calcDirectionCostTo

// Compute cost at this cell and store it in the cell. Also propagates
index field.
//
void yexTable::computeCellCost(int row, int col) {
    yexCell* cell = getCell(row, col);

    int curMinCost = 1000000; // expensive
    yexCell* curMinCell = NULL;
    yexTable_directionT curMinDirec = yexTable_HORIZ;

    // Loop through cells in 3 directions and take min cost one (3way)
    for (yexTable_directionT direc = yexTable_HORIZ; direc <=
yexTable_DIAG;
        direc = (yexTable_directionT) (direc+1)) {
        //int curCost;
        int curCost = directionCostTo(cell, direc);
        if (curCost < curMinCost) {
            curMinCost = curCost;
            curMinCell = predCellInDirection(cell, direc);
            curMinDirec = direc;
        }
    }
    cell->setCost(curMinCost);
    cell->predecessor = curMinCell;
    cell->predDirec = (int) curMinDirec;

    // Propagate index field
    cell->index = curMinCell->index;
} // computeCellCost

void yexTable::dumpTable() {
    static const int HEAD_WIDTH = 13;
    static const int CELL_WIDTH = 9; // was 6 but make room for index
    -shawn 19-Jul-99

    int colsAtaTime = 12;

    // This for loop is for each "clump" of colsAtaTime columns
    for (int colOffset = numLogCols-numPhysCols; colOffset < numLogCols;
colOffset += colsAtaTime) {
        // Write Header -----//
        cerr << endl << endl;
        cerr.width(HEAD_WIDTH);
        cerr << "Column:";

```

```

        //for (int col = numLogCols-numPhysCols; col < numLogCols; col++)
    {
        for (int col = colOffset; col < colOffset + colsAtaTime && col <
numLogCols; col++) {
            cerr.width(CELL_WIDTH);
            cerr << col;
        }
        cerr << endl;
        cerr.width(HEAD_WIDTH);
        cerr << "Token:";
        //for (int col = numLogCols-numPhysCols; col < numLogCols; col++)
    {
        for (int col = colOffset; col < colOffset + colsAtaTime && col <
numLogCols; col++) {
            cerr.width(CELL_WIDTH);
            if (getToken(col)) {
                cerr << getToken(col)->tag();
            } else {
                cerr << "<null>";
            }
        }
        cerr << endl;

        // Write row data -----//

        for (int row = 0; row < numRows; row++) {
            cerr.width(3);
            cerr << row;
            cerr.width(HEAD_WIDTH-3);
            if (row == 0) {
                cerr << "<epsilon>";
            } else {
                OTCLIB_ASSERT(getPatToken(row));
                cerr << getPatToken(row)->tag();
            }
            //for (int col = numLogCols-numPhysCols; col < numLogCols; col++)
        {
            for (int col = colOffset; col < colOffset + colsAtaTime && col <
numLogCols; col++) {
                yutString direcStr = " ";
                yexCell* cell = getCell(row, col);
                if (row > 0 && col >> 0 && getCost(row, col) >= 0 && cell) {
                    if (cell->predDirec == (int) yexTable_HORIZ) {
                        direcStr = " ";
                    } else if (cell->predDirec == (int) yexTable_VERT) {
                        direcStr = "|";
                    } else if (cell->predDirec == (int) yexTable_DIAG) {
                        direcStr = "\\";
                    }
                }
            }

            yutString numStr = yutString::valueOf(getCost(row, col));
            cerr.width(CELL_WIDTH);

            //cerr << direcStr + numStr;

            cerr << direcStr + numStr;

```

```

        //cerr << direcStr + numStr + ":" +
yutString::valueOf(getCell(row, col)->index);

        } // for col
        cerr << endl;
    } // for row
} // for each clump
} // dumpTable()

void yexTable::dumpPattern() {
    cerr << "---- Pattern:" << endl;
    for (int row = 0; row < numRows; row++) {
        dumpCell(getCell(row, numLogCols - 1), false /*showDataTags*/); //
col is arbitrary, but must exist
    }
} // dumpPattern

// If showDataTags then shows tags from input html file.
//
void yexTable::dumpCell(yexCell* cell, bool showDataTags) {
    if (showDataTags) {
        cerr << "(" << cell->col << "," << cell->row << ")" ";
    } else {
        cerr << cell->row << " ";
    }
    //cerr << "(c,r)=( " << cell->col << "," << cell->row << "). PatTag=
";

    yexPatElem* patElem = getPatElem(cell->row);
    yhpHtmlToken* token;

    if (patElem && (token = patElem->tokenP_)) {
        cerr << token->toWellFormedString() << endl;
    } else {
        cerr << "null" << endl;
    }

    if (showDataTags) {
        cerr << " DataTag= ";
        token = getToken(cell->col);
        if (token) {
            cerr << token->toWellFormedString() << endl;
        } else {
            cerr << "null" << endl;
        }
    }

    for (int i = 0; i < (int) patElem->filters_.size(); i++) {
        if (patElem->filters_[i] == NULL)
            cerr << " !filter " << i << " is NULL!";
        else
            cerr << " " << *patElem->filters_[i] << endl;
    }
    for (int i = 0; i < (int) patElem->extractions_.size(); i++) {
        if (patElem->extractions_[i] == NULL)

```

```

        cerr << " !extraction " << i << " is NULL!";
    else
        cerr << " " << *patElem->extractions_[i] << endl;
}

// for (int i = 0; i < (int) extractions_.size(); i++) {
} // dumpCell

void yexTable::setToken(int col, yhpHtmlToken* tokenP) {
    OTCLIB_ASSERT(col >= 0);
    OTCLIB_ASSERT(col >= numLogCols-numPhysCols && col < numLogCols);
    OTCLIB_ASSERT(tokenP); // currently assumes this

    if (false && col == 138) {
        cerr << "settok col138 " << tokenP->type() << endl;
        tokenP->describeToken();
    }

    // If any token there now, delete it.
    int physCol = col % numPhysCols;

    if (tokens__[physCol]) {
        tokens__[physCol]->decrRefCount(); // will delete if last
reference
        tokens__[physCol] = NULL;
    }

    tokens__[physCol] = tokenP;
}

yhpHtmlToken* yexTable::getToken(int col) {
    OTCLIB_ASSERT(col >= 0);
    OTCLIB_ASSERT(col >= numLogCols-numPhysCols && col < numLogCols);

    return tokens__[col % numPhysCols];
}

yhpHtmlToken* yexTable::getPatToken(int row) {
    OTCLIB_ASSERT(row >= 0);
    OTCLIB_ASSERT(row < numRows);

    if (!patElems_[row]) {
        return NULL;
    }
    return patElems_[row]->tokenP;
}

void yexTable::resize(int numRows, int numPhysCols, yexPatElem_Vect
*patElems) {
    OTCLIB_ASSERT(numRows >= (int) patElems->size()+1);

    bool cellsChanged = false;

```

```

for (int col = 0; col < this->numPhysCols; col++) {
    if (tokens__[col]) tokens__[col]->decrRefCount();
    tokens__[col] = NULL;
}

if (numPhysCols >= this->numPhysCols) {
    tokens__.erase(tokens__.begin(), tokens__.end());
    tokens__.reserve(numPhysCols);
    for (int col = 0; col < numPhysCols; col++) {
        tokens__[col] = NULL;
    }
}

for (int i = 0; i < this->numRows; i++) {
    if (i == 0 && patElems_[i]) delete patElems_[0];
    patElems_[i] = NULL;
}

if (numRows >= this->maxRows) {
    patElems_.erase(patElems_.begin(), patElems_.end());
    patElems_.reserve(numRows);
    for(int i = 0; i < numRows; i++) patElems_[i] = NULL;
}

patElems_[0] = new yexPatElem(new yhpHtmlTagToken((yutString)
"<epsilon>"));

for (int i = 1; i < numRows+1; i++) {
    patElems_[i] = (*patElems_)[i-1];
}

if (numRows >= maxRows || numPhysCols >= maxCols) {

    for (int i = 0; i < this->maxRows * this->maxCols; i++) {
        if (cells[i]) delete cells[i];
        cells[i] = NULL;
    }

    maxRows = numRows;
    maxCols = numPhysCols;

    cells.erase(cells.begin(), cells.end());
    cells.reserve(maxRows*maxCols);
    cellsChanged = true;
}

this->numRows = numRows;
this->numPhysCols = numPhysCols;
numLogCols = numPhysCols;

OTCLIB_ASSERT(numRows >= (int) patElems->size()+1);

for (int row = 0; row < numRows; row++) {
    for (int col = 0; col < numPhysCols; col++) {
        yexCell *yexCellP;

```

```

        if (!cellsChanged) yexCellP = getCell(row, col, false);
        else {
            yexCellP = new yexCell(row, col);
            setCell(row, col, yexCellP);
        }
        yexCellP->reset(row, col);
        if (row == 0) {
            yexCellP->setCost(0);
            yexCellP->index = col;
        } else if (col == 0) {
            yexCellP->setCost(row);
        } else {
            yexCellP->setCost(-1); // means not set
        }
    }
}

void yexTable::setAvailable(bool av) {
    available = av;
}

bool yexTable::isAvailable() {
    return available;
}

```

table.h

```
#ifndef yexTable_h
#define yexTable_h
```

```
/**
```

```
 * This class represents the approximate string matching
   table, used in the (dynamic programming) matching algorithm.
   It is implemented as a circular array that wraps around
   on itself and presents an interface of a logical array
   that has many columns (implemented on a physical array
   that has in general fewer columns).
```

```
*/
```

```
#include <stdlib.h>
#include <vector.h>
#include "yex/cell.h"
#include "yex/patElem.h"
```

```
typedef vector<yexCell*> yexTableVect;
```

```
// Represent the directions from a given cell to it's neighbors
// or cost calculation
```

```
//
enum yexTable_directionT {yexTable_HORIZ, yexTable_VERT, yexTable_DIAG};
class yhpHtmlToken;
```

```
class yexTable {
```

```
private:
```

```
    yexTableVect cells; // We do a 2-d array as 1-d and do index xlation
```

```
    // so this has length numRows * numPhysCols
```

```
    vector<yhpHtmlToken*> tokens__; // size is numPhysCols
```

```
    int numRows;
    int numPhysCols; // physical cols
```

```
    int numLogCols; // Max logical column seen so far +1(array
    // logical size grows automatically when
    // higher cols are accessed.)
```

```
    void markColForWrite(int row, int col);
```

```
    int maxRows, maxCols;
    bool available;
```

```
public:
```

```
    // public (lamely) due to yexPatMatcher::writeOutConfig,
    vector<yexPatElem*> patElems; // size is numRows
    int getNumLogCols() { return numLogCols;};
    int getNumPhysCols() { return numPhysCols;};
    yexTable(int numRows, int numPhysCols, yexPatElem_Vect* patElems);
    yexTable(int numRows, int numPhysCols);
```

```

~yexTable();
yexCell* getCell(int row, int col, bool checkCol = true);
void setCell(int row, int col, yexCell* cell);
void shiftTable(int col);
int getCost(int row, int col);
void setCost(int row, int col, int cost);
int directionCostTo(yexCell* cell, yexTable_directionT direc);
void computeCellCost(int row, int col);
yexCell* predCellInDirection(yexCell* cell, yexTable_directionT
direc);
void dumpTable();
void dumpPattern();
void setToken(int col, yhpHtmlToken* tokenP);
yhpHtmlToken* getToken(int col);
void reset();

yhpHtmlToken* getPatToken(int row);
yexPatElem* getPatElem(int row) {return patElems_[row];};
void dumpCell(yexCell* cell, bool showDataTags=true);

void resize(int numRows, int numPhysCols, yexPatElem_Vect *patElems);
bool isAvailable();
void setAvailable(bool av);
int getNumRows() { return numRows; }

}; // class yexTable {
#endif // yexTable_h

```

tablePool.cc

```
#include "tablePool.h"
```

```
const int yexTablePool::ROW_LEN;  
const int yexTablePool::COL_LEN;
```

```
yexTablePool::yexTablePool() {
```

```
    init(10);
```

```
}
```

```
yexTablePool::yexTablePool(int size) {
```

```
    init(size);
```

```
}
```

```
yexTablePool::~yexTablePool() {
```

```
    for(unsigned int i = 0; i < pool.size(); i++) {  
        delete pool[i];
```

```
    }
```

```
}
```

```
void yexTablePool::init(int size) {
```

```
    for(int i = 0; i < size; i++) {  
        pool.push_back(new yexTable(ROW_LEN, COL_LEN));
```

```
    }
```

```
}
```

```
yexTable *yexTablePool::getTable() {
```

```
    for(unsigned int i = 0; i < pool.size(); i++) {  
        if (pool[i]->isAvailable()) {  
            //cout << "returned " << i << endl;
```

```
            return pool[i];
```

```
        }
```

```
    }
```

```
}
```

```
    //cout << "returned new" << endl;
```

```
    yexTable *table = new yexTable(ROW_LEN, COL_LEN);
```

```
    pool.push_back(table);
```

```
    return table;
```

```
}
```

```

tablePool.h

#ifndef yexTablePool_h
#define yexTablePool_h

#include "yex/table.h"

class yexTablePool {
private:
    static const int ROW_LEN = 160;
    static const int COL_LEN = 180;

    vector <yexTable *> pool;

    void init(int size);

public:
    yexTablePool();
    yexTablePool(int size);
    ~yexTablePool();
    yexTable *getTable();
};

#endif

```

test1.cc

```
#include <iostream.h>
#include <fstream.h>
#include "yex/patMatcher.h"
#include "yex/tokFilter.h"
#include "yhp/htmlTagToken.h"
#include "yut/string.h"

int main() {
    //yhpHtmlTagToken::test();

    cerr << "begin yex/test1::main" << endl;
    yexPatMatcher::test();
}

/*
void yexPatMatcher::test() {
    ifstream patternFile("pattern", ios::in);
    ifstream configFile("config", ios::in);
    ifstream filterFile("filter", ios::in);
    ifstream infile("sample.htm", ios::in);

    yexPatMatcher patMatcher;
    patMatcher.setInputStream(infile);

    patMatcher.init(patternFile, configFile, filterFile);

    if (debugLevel_ > 0) {
        patMatcher.tableP_>dumpPattern();
    }

    yutHash resultHash;
    while (patMatcher.nextItem(resultHash)) {
        cerr << "Outputting Match results:" << endl;
        yutStringPairIterator pairs = resultHash.pairs();
        while (pairs.isValid()) {
            cerr << "    Result " << pairs.key() << "=" << pairs.item() <<
endl;
            pairs.next();
        }

        if (debugLevel_ > 2) {
            patMatcher.tableP_>dumpTable();
        }
        resultHash.clear(); // clear hash between calls
    }

} // test()

*/
```

tokFilter.cc

```
#include <fstream.h>
#include <iostream.h>
#include "tokFilter.h"
#include "htmlToken.h"
#include "htmlToker.h"
#include "htmlTagToken.h"

// constructor
yexTokFilter::yexTokFilter() {
    keepCommentToken = false;
    keepTextToken = true;
    setDefaultFilterTokens();
} // constructor

// Return true if token should be filtered out
//
bool yexTokFilter::filterOut(yhpHtmlToken* tokenP) {
    if (tokenP == NULL) return true; // sanity check
    if (tokenP->type() == yhpHtmlToken::COMMENT_TYPE) {
        return ! keepCommentToken;
    }
    if (tokenP->type() == yhpHtmlToken::TEXT_TYPE) {
        return ! keepTextToken;
    }

    if (tokenP->type() == yhpHtmlToken::TAG_TYPE) {
        return filterTokens.contains(((yhpHtmlTagToken*)tokenP)-
>getName());
    }

    // pass through any unknown types (don't filter out)
    return false;
} // filter

void yexTokFilter::addFilterToken(const yutString& tokenName) {
    filterTokens.set(tokenName, "");
} // addFilterToken

void yexTokFilter::removeFilterToken(const yutString& tokenName) {
    filterTokens.remove(tokenName);
} // removeFilterToken

void yexTokFilter::test() {
    ifstream infile("yahoo.htm", ios::in);
    yhpHtmlToker htmlToker(infile);
    yhpHtmlToken* htmlTokP;
    addFilterToken("table");
    removeFilterToken("b");

    while ((htmlTokP = htmlToker.nextToken())) {
        if (filterOut(htmlTokP)) {
            cerr << "filtered: " << htmlTokP->tag() << endl;
        }
    }
}
```

```

    }
} // test

void yexTokFilter::setDefaultFilterTokens() {
    filterTokens.set("!doctype", "");
    filterTokens.set("font", "");
    filterTokens.set("b", "");
    filterTokens.set("br", "");
    filterTokens.set("nobr", "");
    filterTokens.set("map", "");
    filterTokens.set("i", "");
    filterTokens.set("strong", "");
    filterTokens.set("p", "");
    filterTokens.set("tt", "");
    filterTokens.set("h1", "");
    filterTokens.set("h2", "");
    filterTokens.set("h3", "");
    filterTokens.set("h4", "");
    filterTokens.set("h5", "");
    filterTokens.set("h6", "");
    filterTokens.set("form", "");
    filterTokens.set("inset", "");
    filterTokens.set("select", "");
    filterTokens.set("option", "");
    filterTokens.set("textarea", "");

} // setDefaultFilterTokens

```

```

tokFilter.h

#ifndef yexTokFilter_h
#define yexTokFilter_h

/**
** This class implements a token filter used by the pattern
** matcher to filter out useless tokens during pattern matching.
**
**
*/

#include "yut/hash.h"

class yhpHtmlToken;

class yexTokFilter {
private:
    yutHash filterTokens;

public:
    bool keepCommentToken;
    bool keepTextToken;

    // Constructor
    yexTokFilter();

    // Return true if token should be filtered out
    //
    void test();
    bool filterOut(yhpHtmlToken* tokenP);
    void addFilterToken(const yutString& tokenName);
    void removeFilterToken(const yutString& tokenName);
    void setDefaultFilterTokens();

}; // class yexTokFilter {

#endif // yexTokFilter_h

```